PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

Applicant's or agent's file reference 4199-00004	FOR FURTHER ACTION	See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)
International application No. PCT/US99/30584	International filing date *(day/month/year)* 21 DECEMBER 1999	Priority date *(day/month/year)* 19 JANUARY 1999
International Patent Classification (IPC) or national classification and IPC IPC(7): H04B 1/66; G06K 9/36, 9/46 and US Cl.: 375/240; 382/232, 276		
Applicant PENNER, ROBERT C.		

1. This international preliminary examination report has been prepared by this International Preliminary Examining Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 3 sheets.

   ☐ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority. (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

   These annexes consist of a total of 0 sheets.

3. This report contains indications relating to the following items:

   I ☒ Basis of the report

   II ☐ Priority

   III ☐ Non-establishment of report with regard to novelty, inventive step or industrial applicability

   IV ☐ Lack of unity of invention

   V ☒ Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

   VI ☐ Certain documents cited

   VII ☐ Certain defects in the international application

   VIII ☐ Certain observations on the international application

Date of submission of the demand 14 JULY 2000	Date of completion of this report 08 SEPTEMBER 2000
Name and mailing address of the IPEA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer STEPHEN CHIN Telephone No. (703) 306-4713

Form PCT/IPEA/409 (cover sheet) (July 1998) ☆

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/US99/30584

I. Basis of the report

1. With regard to the elements of the international application:*

[X] the international application as originally filed

[X] the description:
pages 1-73 , as originally filed
pages NONE , filed with the demand
pages NONE , filed with the letter of _____

[X] the claims:
pages 74-96 , as originally filed
pages NONE , as amended (together with any statement) under Article 19
pages NONE , filed with the demand
pages NONE , filed with the letter of _____

[X] the drawings:
pages 1-14 , as originally filed
pages NONE , filed with the demand
pages NONE , filed with the letter of _____

[X] the sequence listing part of the description:
pages NONE , as originally filed
pages NONE , filed with the demand
pages NONE , filed with the letter of _____

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.
These elements were available or furnished to this Authority in the following language _____ which is:

[ ] the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).

[ ] the language of publication of the international application (under Rule 48.3(b)).

[ ] the language of the translation furnished for the purposes of international preliminary examination (under Rules 55.2 and/or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

[ ] contained in the international application in printed form.

[ ] filed together with the international application in computer readable form.

[ ] furnished subsequently to this Authority in written form.

[ ] furnished subsequently to this Authority in computer readable form.

[ ] The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

[ ] The statement that the information recorded in computer readable form is identical to the writen sequence listing has been furnished.

4. [X] The amendments have resulted in the cancellation of:

[X] the description, pages NONE

[X] the claims, Nos. NONE

[X] the drawings, sheets/~~fig~~ NONE

5. [ ] This report has been drawn as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

** Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as "originally filed" and are not annexed to this report since they do not contain amendments (Rules 70.16 and 70.17).*

***Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.*

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/US99/30584

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. statement

Novelty (N)	Claims	1-65	YES
	Claims	None	NO
Inventive Step (IS)	Claims	1-65	YES
	Claims	None	NO
Industrial Applicability (IA)	Claims	1-65	YES
	Claims	None	NO

2. citations and explanations (Rule 70.7)

Claims 1-65 meet the criteria set out in PCT Article 33(2)-(4), because the prior art does not teach or fairly suggest a method of compressing and reconstructing input data comprising the steps of sampling the input at the points of the Farey quadrature; transforming the Farey-sampled data into an intermediate form; quantizing the Farey-sampled data while in the intermediate form; using the quantized intermediate form to store or transmit the Farey-sampled data; de-quantizing the quantized intermediate form of the Farey-sampled data prior to reconstructing the data; and reconstructing the intermediate form of the Farey-sampled data in order to obtain reconstructed data for use.

---------- NEW CITATIONS ----------
NONE

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 :
H04B 1/66, G06K 9/36, 9/46

A1

(11) International Publication Number: WO 00/44104

(43) International Publication Date: 27 July 2000 (27.07.00)

(21) International Application Number: PCT/US99/30584

(22) International Filing Date: 21 December 1999 (21.12.99)

(30) Priority Data:
60/116,540 19 January 1999 (19.01.99) US

(71)(72) Applicant and Inventor: PENNER, Robert, C. [US/US]; 528 Fifth Street, Manhattan Beach, CA 90266 (US).

(74) Agents: WILLIAMS, Edward, R., Jr. et al.; Andrus, Sceales, Starke & Sawall, LLP, Suite 1100, 100 East Wisconsin Avenue, Milwaukee, WI 53202 (US).

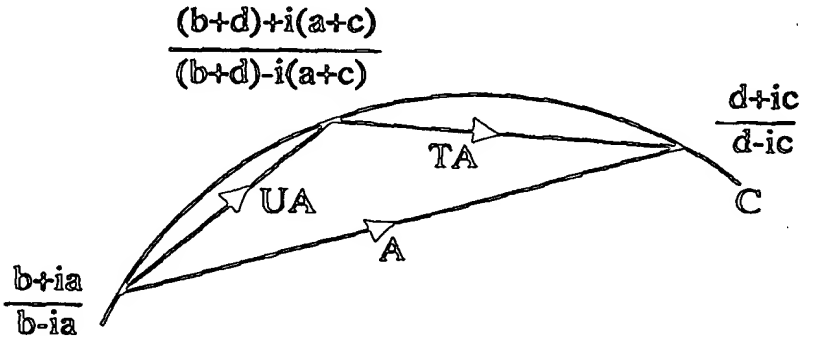
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHODS OF DIGITAL FILTERING AND MULTI-DIMENSIONAL DATA COMPRESSION USING THE FAREY QUADRATURE AND ARITHMETIC, FAN, AND MODULAR WAVELETS

(57) Abstract

Methods are presented for calculating the wavelet filters and their inverses which rely on a new method for sampling (UA, TA, A) either digital or analog data. These methods combine and extend to give novel procedures for non-reversible multi-dimensional data compression. For selected applications, this procedure improves achievable compression factors by an estimated one to three orders of magnitude and is well-suited to picture build-up or other iterative refinement. Combining these wavelet filters and their inverses with previous theoretical work furthermore provides novel methods for calculating Fourier and other transforms. In a preferred embodiment used to calculate the Fourier transform (1) and its inverse (34) applied to digital input data, the method replaces the fast Fourier transform and its inverse and provides improvements in achievable accuracy. The new sampling method is inherently multiscale, and the invention thereby obviates the usual Nyquist constraint on the meaningful bandwidth in terms of the number of samples. Finally, the invention provides novel and efficient analog-to-digital and digital-to-analog interface.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHODS OF DIGITAL FILTERING AND MULTI-DIMENSIONAL DATA COMPRESSION USING THE FAREY QUADRATURE AND ARITHMETIC, FAN, AND MODULAR WAVELETS

Field of the Invention

It is well-known that digital filtering of digitally represented samples of analog data can be simplified and improved by performing the required processing in some transformed domain. Using both a transform and its inverse to simplify the digital or other manipulation of data, one first transforms the data from its initial form to an intermediate form, then applies a specified manipulation, and finally applies the inverse transform to recover the data in its initial form from its intermediate form. By way of example and not by way of limitation: the initial form of the data may be digital samples of analog data, and the intermediate form of the data may be the coefficients of the Fourier transform; the initial form of the data may be the coefficients of the Fourier transform, and the intermediate form of the data may be the coefficients in some wavelet or other expansion; the initial form of the data may be an analog signal, and the intermediate form of the data may be some digital representation of it; the initial form of the data may be digital samples of an analog signal, and the intermediate form of the data may be a quantization of the digital samples. For instance, transform coding methods for data compression in signal or data processing are realized in practice as transform of data/quantization, storage or transmission, de-quantization/reconstruction. Another class of examples consists of finite-impulse response digital filters where the input data is filtered using the indirect calculation of convolution given by Fourier transform/multiplication/inverse Fourier transform. For analog data, sampling and filtering together determine fidelity and speed of manipulation.

The invention disclosed here provides a new method for sampling analog or digital input data which may be used to calculate new wavelet transforms as well as their inverse reconstruction algorithms. This immediately provides novel and efficient methods for data compression based on this new method of non-linear transform coding. In combination with these wavelet filters, the invention also provides new methods for calculating various classical transforms including the Fourier transform and its inverse. This immediately provides novel and efficient methods for digital filtering. By way of example and not of limitation, practical applications of the methods include non-speech audio compression, speech compression, speech recognition, speech synthesis, voice printing, audio filtering for hearing aids, still two-dimensional image compression, moving video compression, video compression for purposes of telephony, precision Fourier analysis, precision trigonometry, denoising, interpolation, medical imaging, geological imaging for recovery of oil or other resources, and other emission/detection apparatus. In specific applications, it may be necessary to store streams of input data in a buffer for separate processing or manipulation while concurrently collecting further input data whether analog or digital. It may also be useful in practice in particular applications to calculate and archive or otherwise store specific coefficients or constants and recover them from memory at run time rather than computing them at run time.

Description of Prior Art

In the landmark paper "Orthonormal bases of compactly supported wavelets, *Communications in Pure and Applied Mathematics* 41 , pp. 909-996 (1988), Daubechies

presented a class of w elets which have been effectively applie· ı numerous contexts; see, for instance, US Patents 5,453,945 by Tucker et al., 5,606,575 by Williams, as well as Daubechies' monograph "Ten Lectures on Wavelets" SIAM (1992) and its references. In another landmark paper, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of Computation* **19**, pp. 297-301 (1965), J. W. Cooley and J. W. Tukey described a simplified algorithm for the calculation of Fourier series on the computer. The utility of this fast Fourier transform FFT is well known with thousands of important applications and implementations; see, for instance, US Patents 3,871,577 by Avellar et al., 3,881,100 by Works et al., 4,051,357 by Bonnerot, and "The DFT, an owner's manual for the discrete Fourier transform", SIAM (1995) by W. Briggs and Van Emden Henson and its references. A requirement for both the FFT method and for the multiscale filtering of the Daubechies' wavelets is that the sampling of input data must be equally spaced though further methods of adaptive refinement of grids have also been developed in each context.

Other examples of transforms whose efficient digital implementations are important and to which the invention is relevant include the transforms of Hilbert, Haar, Laplace, Bessel, Laguerre, Hermite, Chebyshev, Hotelling, Mersenne, and Fermat; see, for instance, US Patents 3,891,443 by Lynch et al. and 4,093,994 by Nussbaumer.

Invention Summary

Three new families of wavelets, respectively called *arithmetic wavelets*, *fan wavelets*, and *modular wavelets*, are defined on the standard circle of radius one in the complex plane. Any complex-valued function defined on this circle may be written essentially uniquely as a complex linear combination of each family, and the approximation of specified input data as a finite linear combination for each family leads to a corresponding wavelet filter. Each filter depends upon a new quadrature on the circle, called the *Farey quadrature*, which relies on a novel method of non-equally-spaced sampling. The output of each wavelet filter plays the role in the invention of an intermediate representation between input data and any one of a number of useful output transformations of data which may be computed from the intermediate quantity.

With either analog or digital input data, it is convenient to think of the given input data as spatial data. The inverse wavelet transform or *reconstruction algorithm* requires calculating spatial data from wavelet data and admits an especially convenient implementation: the values of the spatial function at certain non-equally-spaced grids of arbitrarily fine spacing may be computed exactly using a specified finite set of wavelet coefficients. This reconstruction algorithm combines with the wavelet filter into a binary cascade giving an efficient procedure for data compression which is especially well-suited to discontinuous input data and to iterative refinement of output data. Furthermore, the methods extend directly to procedures for compression of multi-dimensional data. Computer coding for the reconstruction algorithm itself is sufficiently abbreviated and low-level that it might be transmitted together with compressed data, for instance, for down-loading from satellite to home computer.

For the calculation of classical transforms of spatial input data, it is convenient to think of the desired output data, for instance, the Fourier coefficients, as frequency data. The transform from spatial to frequency data is accomplished in two main steps, the first of which approximates the input data by wavelets using the wavelet filter. The second

step is the calculation Fourier coefficients or other frequency dat in terms of wavelets and requires formulas derived in previous theoretical work. Similarly, the calculation of the inverse transform depends upon the reconstruction algorithm and known formulas which express frequency data in terms of the wavelets. The calculation of a transform or its inverse admits an elegant implementation as a binary cascade.

Brief Description of Figures

FIG 1a illustrates the standard circle C of radius one in the complex plane, where C bounds the standard disk D of radius one. The complex numbers $+1$ and -1 are also indicated, and the chord of C with these endpoints is labeled by the matrix $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

FIG 1b illustrates a chord in the top semi-circle of C labeled by a matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, together with two further "descendant" chords of C labeled by the matrix products UA and TA as indicated, where $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. The three chords determine a triangle inscribed in C, and the vertices of this triangle are indicated as complex numbers.

FIG 1c illustrates a chord in the bottom semi-circle of C labeled by a matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, together with two further "descendant" chords of C labeled by the matrix products $U^{-1}A$ and $T^{-1}A$ as indicated, where $U^{-1} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$ and $T^{-1} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$. The three chords determine a triangle inscribed in C, and the vertices of this triangle are indicated as complex numbers.

FIG 2 depicts a classical figure in mathematics called "the Farey tesselation in Klein's model of hyperbolic geometry". It is constructed beginning from the chord in Figure 1a by recursively taking descendants as in Figure 1b and Figure 1c, respectively, on the top and bottom semi-circle of C. The figure itself is classical, but the sampling of input data on C at the endpoints of the chords of the Farey tesselation in increasing generation is a novel aspect of the invention disclosed here.

FIG 3a illustrates the circle C, the chord labeled I and the two triangles in the Farey tesselation on either side of this chord. The vertices of these triangles are indicated inside the circle as complex numbers. Outside the circle are given four explicit combinations of cosine and sine functions, one such function next to each circular arc determined by the vertices, where θ is the usual angular coordinate on the circle. A function $\tilde{\vartheta}_I(\theta)$ is uniquely determined by the figure, where on each such circular arc, $\tilde{\vartheta}_I(\theta)$ takes the values of the nearby combination of cosine and sine functions. The other parts Figures 3b-3e likewise determine analogous functions $\tilde{\vartheta}_A(\theta)$ for other matrices $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. These functions taken together form the system of wavelets on which the invention is based, and they are unique to this invention.

FIG 3b illustrates th. ıord in the top semi-circle of C labeled th the matrix A, where $a > c$. As before, the figure determines a function on the circle denoted $\tilde{\vartheta}_A(\theta)$.

FIG 3c illustrates the chord in the top semi-circle of C labeled with the matrix A, where $c >= a$. As before, the figure determines a function on the circle denoted $\tilde{\vartheta}_A(\theta)$.

FIG 3d illustrates the chord in the bottom semi-circle of C labeled with the matrix A, where $a > -c$. As before, the figure determines a function on the circle denoted $\tilde{\vartheta}_A(\theta)$.

FIG 3e illustrates the chord in the bottom semi-circle of C labeled with the matrix $A =$, where $-c >= a$. As before, the figure determines a function on the circle denoted $\tilde{\vartheta}_A(\theta)$.

FIG 4 illustrates the notation near the chord labeled by the matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ in the top semi-circle of C; this notation will be useful in several subsequent discussions.

FIG 5 provides a flow chart for the main driving routine in calculating the Fourier transform.

FIG 6 provides a flow chart for the main recursive routine in calculating the Fourier transform.

FIG 7 provides a flow chart for the procedure of updating coefficients in calculating the Fourier transform.

FIG 8 provides a flow chart for the procedure of processing terminal arrows of the recursion in calculating the Fourier transform.

FIG 9 provides a flow chart for the main driving routine in calculating the inverse Fourier transform.

FIG 10 provides a flow chart for the main recursive routine in calculating the inverse Fourier transform.

FIG 11 illustrates the region $\mathcal{U}$ consisting of all complex numbers with positive imaginary part. Map the disk D to $\mathcal{U}$ via the function $z \mapsto i(z+1)/(z-1)$; if two points in C are the endpoints of a chord in Klein's model of the Farey tesselation as in Figure 2, then draw the semi-circle in $\mathcal{U}$ passing through the corresponding two points which is perpendicular to the real axis. This produces another classical figure in mathematics, called the "Farey tesselation in the upper half-space model of hyperbolic geometry", which is indicated in Figure 11.

Description of Preferred Embodiments

In order to disclose the methods, it is necessary to develop some notation and terminology. To this end, consider the standard disk D of radius one about the origin in the complex plane with its usual complex coordinate $z = x + iy$, where $i = \sqrt{-1}$ and x, y are real numbers, together with its boundary circle C as illustrated in Figure 1a. An *arrow* is by definition an oriented chord of the circle C labeled by a two-by-two matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$, where the initial point of the chord is given by the complex number

$\frac{b+ia}{b-ia} \in C$, and the final point of the chord is given by $\frac{d+ic}{d-ic} \in C$. A special arrow called the *doe* (short for "distinguished oriented edge") is given by the diameter of C with endpoints $-1, +1$ labeled by the matrix $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ as in Figure 1a. Also required subsequently are the matrices $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ and $U^n = \begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix}$, $T^n = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}$, for any integer n, which satisfy the identities $S^2 = -I$, $SUS = -T^{-1}$, $STS = -U^{-1}$ of matrix products.

Inductively define two separate binary cascades, one of *top arithmetic arrows* and another of *bottom arithmetic arrows*, as follows:

Basis: The doe, which is illustrated in Figure 1a, is both a top arithmetic arrow and a bottom arithmetic arrow.

Top Induction: As in Figure 1b, given the top arithmetic arrow labeled by $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, there are also top arithmetic arrows labeled by

$$UA = \begin{pmatrix} a & b \\ a+c & b+d \end{pmatrix} \text{ and } TA = \begin{pmatrix} a+c & b+d \\ c & d \end{pmatrix}.$$

Bottom Induction: As in Figure 1c, given the bottom arithmetic arrow labeled by $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, there are also bottom arithmetic arrows labeled by

$$U^{-1}A = \begin{pmatrix} a & b \\ c-a & d-b \end{pmatrix} \text{ and } T^{-1}A = \begin{pmatrix} a-c & b-d \\ c & d \end{pmatrix}.$$

It is a classical theorem in mathematics that the chords underlying any two arithmetic arrows meet only at their endpoints, if at all, and that the family of all such chords decomposes the disk D into an infinite collection of triangles with vertices in the circle C as is illustrated in Figure 2; this figure is called "the Farey tesselation in Klein's model of hyperbolic geometry". There is no suitable reference for this procedure in the literature, and it is for completeness that the explicit recursive definition of arithmetic arrows has been given here. An arithmetic arrow is said to be of *generation g* if it arises from g applications of the inductive clause starting from the doe in case of either top or bottom arrows; the doe has generation zero by convention.

Consider the infinite set $Q \subseteq C$ consisting of the endpoints of chords underlying all arithmetic arrows, and say a point of Q is of *generation g* if it is the endpoint of the chord underlying an arrow of generation g and no less; -1 and $+1$ are of generation zero by convention. There is a canonical enumeration of the generation g points of Q clockwise in C starting from -1 and the associated enumeration of Q itself by increasing generation. For example, the ordering begins with:

$$-1,\ +1,\ +i,\ -i,\ \frac{-1-2i}{-1+2i},\ \frac{-2-i}{-2+i},\ \frac{2-i}{2+i},\ \frac{1-2i}{1+2i},\ \frac{-1-3i}{-1+3i},\ \ldots.$$

Given input data on the circle C, the sampling of it at the points $Q \subseteq C$ in this order of increasing generation will be referred to as the *Farey quadrature*. In practice, one

might interpolate given ta as required at the sampling points of tl Farey quadrature or restrict the Farey quadrature to a circular arc in C. (In the subsequent discussion of mathematical basis, the sample points of the Farey quadrature will be seen to correspond to the rational points in the real line enumerated in increasing order, where the generation is the length of a corresponding continued fraction expansion.)

Notice that the doe separates the triangle with vertices $-1, +1, -i$ from the triangle with vertices $-1, +1, +i$. In the same way, an arbitrary arithmetic arrow separates two triangles complementary to all the arithmetic arrows in D. The endpoints of chords of these triangles in various cases are illustrated in Figures 3a-3e as functions of the matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ labeling the arithmetic arrow. Figure 3a depicts the doe itself with $A = I$, Figure 3b and 3c depict top arrows with $a > c$ and $c \geq a$, respectively, and Figure 3d and 3e depict bottom arrows with $a > -c$ and $-c \geq a$, respectively. The notation inside the circles indicates the endpoints of the chords as complex numbers, and the notation outside the circles in each case of Figures 3a-3e is yet to be explained.

To this end, define a *trigonometric function* to be a real-valued function

$$f(\theta) = \alpha \cos \theta + \beta \sin \theta + \gamma,$$

where θ is the usual angular coordinate on the unit circle C and α, β, γ are real numbers. Next, define a *piecewise trigonometric function* on the circle to be a function $f(\theta)$ so that there is a finite collection of circular arcs $I_j \subseteq C$, for $j = 1, \ldots, J$, any two of which meet at common endpoints in C, if at all, together with a collection of trigonometric functions $t_j(\theta)$ for $j = 1, \ldots, J$, so that $f(\theta) = t_j(\theta)$ if $\theta \in I_j$. In other words, C is decomposed into a finite number of non-overlapping circular arcs, and on each such arc, f takes the values of a trigonometric function.

There is a particular family of piecewise trigonometric functions, one such *arithmetic wavelet* $\tilde{\vartheta}_A(\theta)$ for each matrix A labeling an arithmetic arrow. These functions are defined in Figures 3a-3e, where the endpoints of the circular arcs are as described before, and each trigonometric function is written outside the circle next to its corresponding circular arc. There are thus five cases in the definition of arithmetic wavelet corresponding to the five cases in Figures 3a-3e already discussed, and in each case, the arithmetic wavelet itself is a piecewise trigonometric function defined using exactly four circular arcs. Arithmetic wavelets are a fundamental new ingredient of this invention and no doubt seem entirely ad hoc and unmotivated at this juncture of the discussion. The mathematical basis for them will be given later, but it is worth pointing out now that each arithmetic wavelet $\tilde{\vartheta}_A(\theta)$ is a once-continuously differentiable function taking value zero at the points $-1, +1, -i$ of the circle C, that is, for $\theta = 0, -\pi/2, -\pi$, except for $\tilde{\vartheta}_{U^{-1}}(\theta)$ and $\tilde{\vartheta}_{T^{-1}}(\theta)$ which take the value zero only at the points ± 1.

In order to explicate the definition of arithmetic wavelets, notice that any two-by-two matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ determines a self-mapping $M_A : C \to C$ of the circle C as follows:

$$M_A\left(\frac{t-i}{t+i}\right) = \frac{(ta+b) - i(tc+d)}{(ta+b) + i(tc+d)},$$

and in fact, the image of the endpoints, -1 and $+1$, of the doe under M_A are the endpoints of the arithmetic arrow with label A. Construct from the function $\tilde{\vartheta}_I(\theta)$

defined on C in Figure 1. The corresponding vector field on C given $\tilde{\vartheta}_I(\theta)\,\frac{\partial}{\partial\theta}$, where $\frac{\partial}{\partial\theta}$ is the usual unit tangent vector field to C.

There is an explicit formula to be given presently for the transformation of suitable vector fields under the change of coordinates on C given by M_A. Namely, the vector field $t(\theta)\frac{\partial}{\partial\theta} = [\alpha\cos\theta+\beta\sin\theta+\gamma]\frac{\partial}{\partial\theta}$ transforms under M_A to the vector field $t^A(\theta)\frac{\partial}{\partial\theta}$, where

$$t^A(\theta) = \left\{ \begin{array}{rl} & \frac{1}{2}\,[2(cd-ab)\beta + (d^2-b^2)(\alpha+\gamma) + (a^2-c^2)(\alpha-\gamma)]\cos\theta \\ + & \quad [(ad+bc)\beta + bd(\alpha+\gamma) - ac(\alpha-\gamma)]\sin\theta \\ + & \frac{1}{2}\,[2(cd+ab)\beta + (d^2+b^2)(\alpha+\gamma) - (a^2+c^2)(\alpha-\gamma)] \end{array} \right\}.$$

More generally, given a piecewise trigonometric function $f(\theta)$ taking the values of the trigonometric function $t_j(\theta)$ on the circular arc I_j as before, the corresponding vector field $f(\theta)\frac{\partial}{\partial\theta}$ transforms under M_A to the vector field which on the circular arc $M_A(I_j)$ is given by $t_j^A(\theta)\frac{\partial}{\partial\theta}$.

Changing coordinates on C by M_A using this formula transforms the vector field $\tilde{\vartheta}_I(\theta)\,\frac{\partial}{\partial\theta}$ to another vector field $\vartheta_A(\theta)\frac{\partial}{\partial\theta}$, thereby defining the function $\vartheta_A(\theta)$ on C. Finally except for $A = U^{-1}$ and $A = T^{-1}$, the corresponding arithmetic wavelet is given by

$$\tilde{\vartheta}_A(\theta) = \vartheta_A(\theta) - [\alpha\cos\theta + \beta\sin\theta + \gamma],$$

where α, β, γ are chosen so that $\tilde{\vartheta}(\pi) = \tilde{\vartheta}(0) = \tilde{\vartheta}(-\pi/2) = 0$; the two special cases are

$$\tilde{\vartheta}_{U^{-1}}(\theta) = \vartheta_{U^{-1}}(\theta) + 2\sin\theta \quad \text{and} \quad \tilde{\vartheta}_{T^{-1}}(\theta) = \vartheta_{T^{-1}}(\theta) - 2\sin\theta$$

(which are defined in this manner to guarantee a symmetry of the upper and lower semi-circles of C). This is the abstract definition of arithmetic wavelets which is made explicit in Figure 3.

It is furthermore convenient to define

$$\overline{\vartheta}_A(\theta) = \left\{ \begin{array}{ll} \tilde{\vartheta}_A(\theta); & \text{if } A \neq U^{-1}, T^{-1}, \\ \vartheta_{U^{-1}}(\theta); & \text{if } A = U^{-1}, \\ \vartheta_{T^{-1}}(\theta); & \text{if } A = T^{-1}, \end{array} \right.$$

so that for every label A on an arithmetic arrow, $\overline{\vartheta}_A(\pi) = \overline{\vartheta}_A(0) = \overline{\vartheta}_A(-\pi/2) = 0$.

The arithmetic wavelets enjoy an important *renormalization property*, namely,

$$\overline{\vartheta}_{BA}(M_A(\theta)) \;\; = \;\; M_A'(\theta)\;\overline{\vartheta}_B(\theta),$$

where the prime denotes the usual derivative and where it is required that either both matrices B and A are matrix products of U^{+1}, T^{+1} or both matrices B and A are matrix products of U^{-1}, T^{-1}.

Just as a function $f = f(\theta)$ may be expressed in its representation as a Fourier series $f \sim \sum_n c_n e^{in\theta}$, so too it may be expressed as a series $f \approx \sum e_A\tilde{\vartheta}_A(\theta)$ of arithmetic wavelets, which together form a linearly independent set. The sum is over all arithmetic arrows, and the coefficients e_A are called the *arithmetic wavelet coefficients*. The calculation

of arithmetic wavelet c ficients from the input data f is called th *rithmetic wavelet filter*. The arithmetic wavelet coefficients are not quite uniquely determined by f as will be explained.

The reconstruction algorithm or inverse wavelet transform provides a method for calculating function values at points of the circle from arithmetic wavelet coefficients. To determine the value of the sum $\sum e_A \tilde{\vartheta}_A$ at a specified point $q \in Q \subseteq C$, draw a line segment from q to the origin in D; this segment is contained in a finite collection of triangles complementary to all arithmetic arrows, whose boundary edges taken together form a finite set of chords; the corresponding finite set of arithmetic arrows A enumerates the only coefficients e_A which affect the value of $\sum_{A \in S} e_A \tilde{\vartheta}_A$ at q. This is an important *finiteness property* of arithmetic wavelets which is crucial to the efficiency of the reconstruction algorithm.

This completes the general discussion of arithmetic wavelets, the first family of new wavelets disclosed here, and the two further families of new wavelets are next defined.

For each label A on an arithmetic arrow, there are two *fan wavelets*

$$\phi_A(\theta) = \sum_{n \geq 0} \overline{\vartheta}_{U^n A}(\theta),$$
$$\phi_{SA}(\theta) = \sum_{n \geq 0} \overline{\vartheta}_{T^{-n} A}(\theta),$$

as well as two *modular wavelets*

$$\psi_A(\theta) = \sum_{n \geq 0} n\, \overline{\vartheta}_{U^n A}(\theta),$$
$$\psi_{SA}(\theta) = \sum_{n \geq 0} n\, \overline{\vartheta}_{T^{-n} A}(\theta).$$

It follows immediately from the definitions that there are the following relations among the wavelets for any label A on an arithmetic arrow.

$$\begin{aligned} \phi_A(\theta) &= \psi_{U^{-1}A}(\theta) - \psi_A(\theta), \quad \phi_{SA}(\theta) = \psi_{U^{-1}SA}(\theta) - \psi_{SA}(\theta), \\ \overline{\vartheta}_A(\theta) &= \phi_A(\theta) - \phi_{UA}(\theta) = \phi_{SA}(\theta) - \phi_{USA}(\theta) \\ &= \psi_{U^{-1}A}(\theta) - 2\psi_A(\theta) + \psi_{UA}(\theta) = \psi_{U^{-1}SA}(\theta) - 2\psi_{SA}(\theta) + \psi_{USA}(\theta). \end{aligned}$$

On the other hand as is not at all obvious from the definitions, ϕ_I and ψ_I are given by the following explicit formulas as piecewise trigonometric functions

$$\phi_I = \begin{cases} 2 \sin \theta; & \text{if } 0 \leq \theta \leq \pi, \\ -2 \cos \theta - 2 \sin \theta - 2; & \text{if } \pi \leq \theta \leq 3\pi/2, \\ -2 \cos \theta + 2 \sin \theta + 2; & \text{if } 3\pi/2 \leq \theta \leq 2\pi, \end{cases}$$

and

$$\psi_I(\theta) - = \begin{cases} 2 - 2 \cos \theta; & \text{if } 0 \leq \theta \leq \pi, \\ 0; & \text{if } \pi \leq \theta \leq 2\pi. \end{cases}$$

In fact, modular wavelets arise from ψ_I in exactly the same manner that arithmetic wavelets arise from $\tilde{\vartheta}_I$, namely, $\psi_I(\theta)\frac{\partial}{\partial \theta}$ transforms under M_A to the vector field

$\varphi_A^*(\theta)\frac{\partial}{\partial\theta}$, and $\psi_A(\theta) = \quad_1(\theta) - [\alpha \cos\theta + \beta \sin\theta + \gamma]$, where α, γ are chosen to guarantee that $\psi_A(0) = \psi_A(\pi) = \psi_A(-\pi/2) = 0$; transforming $\psi_I(\theta)\frac{\partial}{\partial\theta}$ using the change of coordinates given by M_{SA} likewise gives rise to $\psi_{SA}(\theta)$. The fan wavelets also arise from ϕ_I in the analogous manner.

In contrast to arithmetic wavelets, neither the fan nor modular wavelets are linearly independent, and in each case, there is one linear relation for each arithmetic arrow, namely,

$$\begin{aligned} \phi_A - \phi_{UA} &= \phi_{SA} - \phi_{USA}, \\ \psi_{U^{-1}A} - 2\psi_A + \psi_{UA} &= \psi_{U^{-1}SA} - 2\psi_{SA} + \psi_{USA}. \end{aligned}$$

On the other hand, the following collection of modular wavelets forms a basis

$$\begin{aligned} &\{\psi_A : A \text{ labels a top arithmetic arrow and } c \geq a\} \\ \cup\ &\{\psi_{SA} : A \text{ labels a top arithemtic arrow and } a > c\} \\ \cup\ &\{\psi_A : A \text{ labels a bottom arithemtic arrow and } -c \geq a\} \\ \cup\ &\{\psi_{SA} : A \text{ labels a bottom arithemtic arrow and } a > -c\}; \end{aligned}$$

there is an analogous basis for the fan wavelets.

Both fan and modular wavelets enjoy precisely the same finiteness property as arithmetic wavelets. Furthermore, both fan and modular wavelets enjoy renormalization properties, namely,

$$\begin{aligned} \phi_{BA}(M_A(\theta)) &= M_A'(\theta)\ \phi_B(\theta), \\ \psi_{BA}(M_A(\theta)) &= M_A'(\theta)\ \psi_B(\theta), \end{aligned}$$

where the requirements on A, B are as before.

This completes the necessary abstract definitions. The methods will first be presented using arithmetic wavelets for real-valued input functions with extensions to modular and fan wavelets and complex-valued or multi-dimensional input/output data to be described later. As a point of notation throughout these discussions, there will be the tacit assumption that the entries in the matrix denoted A are given by $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

The Fourier Transform

Included in the section entitled "Source Codes" is an implementation in the computer language C of a preferred embodiment of the method described in this section employing arithmetic wavelets.

Suppose that $f(\theta)$ is some specified real-valued input function defined on the circle. The calculation of the Fourier transform of f proceeds by two main steps:

Step1 : Choose a finite set S of arithmetic arrows and approximate $f \approx \sum_{A\in S} e_A \tilde{\vartheta}_A$ calculating the coefficients e_A in a manner to be described in terms of the values of f at specified points using the Farey quadrature.

Step 2: Substitute into the expression in Step 1 the known Fourier expansions $\tilde{\vartheta}_A(\theta) \sim \sum_n c_n^A\ e^{in\theta}$ in order to derive the approximation

$$c_n = \sum_{A\in S} e_A\ c_n^A.$$

The coefficients require n Step 2 are known theoretically and giv for $n \neq 0, \pm 1$ by

$$\begin{aligned}\pi\, i\, (n^3 - n)\ c_n^A \ = \ & -[(c-a)^2 + (b-d)^2]\left[\frac{(b-d) - i(a-c)}{(b-d) + i(a-c)}\right]^n \\ & + 2(c^2+d^2)\left[\frac{d-ic}{d+ic}\right]^n \ + \ 2(a^2+b^2)\left[\frac{b-ia}{b+ia}\right]^n \\ & - [(c+a)^2 + (b+d)^2]\left[\frac{(b+d) - i(a+c)}{(b+d) + i(a+c)}\right]^n.\end{aligned}$$

Furthermore, the coefficients c_0^A, c_1^A, c_{-1}^A are given by

$$\begin{aligned}\pi c_{\pm 1}^A = \ & \theta_h(c^2 - d^2 \pm 2icd) \ + \ \theta_r[(b-d)^2 - (a-c)^2 \mp 2i(b-d)(a-c)]/2 \\ & + \ \theta_t(a^2 - b^2 \pm 2iab) \ + \ \theta_\ell[(b+d)^2 - (a+c)^2 \mp 2i(b+d)(a+c)]/2,\end{aligned}$$

$$\begin{aligned}\pi c_0^A = \ & 2\theta_h(c^2+d^2) \ - \ \theta_r[(b-d)^2 + (a-c)^2] \\ & + 2\theta_t(a^2+b^2) \ - \ \theta_\ell[(b+d)^2 + (a+c)^2],\end{aligned}$$

where the angles are

$$\theta_h = \tan^{-1}\left(\frac{2cb}{d^2-c^2}\right), \quad \theta_\ell = \tan^{-1}\left(\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}\right),$$

$$\theta_t = \tan^{-1}\left(\frac{2ab}{b^2-a^2}\right), \quad \theta_r = \tan^{-1}\left(\frac{2(b-d)(a-c)}{(b-d)^2-(a-c)^2}\right).$$

It may be useful in practice in particular applications to calculate and archive or otherwise store the constants required in the previous expressions for c_n^A and recover them from memory at run time rather than computing them at run time.

The Fourier coefficients $c_0, c_{\pm 1}$ of f thus require special treatment which does not present additional challenges. For simplicity, suppose that the input function $f(\theta)$ has these three coefficients fixed by the condition that $f(\theta) = 0$ for $\theta = 0, -\pi/2, -\pi$. More precisely, if $f(\theta)$ is any real-valued function on the circle, then let $\overline{f}(\theta)$ denote its *normalization* defined by $\overline{f}(\theta) = f(\theta) + \alpha \cos\theta + \beta \sin\theta + \gamma$, where α, β, γ are chosen so that $\overline{f}$ takes value zero at $0, -\pi/2, -\pi$. The specified simplifying assumption on f thus amounts to the condition that $f = \overline{f}$.

The procedure for calculating wavelet coefficients is slightly different depending upon whether the corresponding arithmetic arrow is a top or bottom arrow. However, the antipodal map $\theta \mapsto \pi + \theta$ transforms the bottom of the circle C to the top in such a way that it suffices to discuss only the case of top arrows. More precisely, the algorithm for bottom arrows arises from that for top arrows by:

- replacing the input data value $f(\theta)$ with the input data value $f(\theta + \pi)$,
- replacing the matrix A in the described manipulations for top arrows by the conjugate matrix SAS for bottom arrows, and
- replacing the trigonometric function $\alpha\cos\theta + \beta\sin\theta + \gamma$ for top arrows by the trigonometric function $-\alpha\cos\theta - \beta\sin\theta + \gamma$ for bottom arrows.

The algorithm and computer coding for bottom arrows is therefore derived without difficulty from that for top arrows, and the subsequent discussion is specialized without loss of generality to top arrows.

In addition to the function $f(\theta)$ defined on the circle, suppose that there is also specified a bandwidth of interest N (i.e. the method should calculate the Fourier coefficients c_n only in the range $|n| \leq$ N) and a tolerance EPS (where contributions to the specified Fourier coefficients are regarded as negligible if they are of magnitude less than EPS). There are numerous variants of the method which depend upon other parameters which may be tailored to a given application, and some these variants will be discussed further.

Steps 1 and 2 are merged into a single binary cascade which keeps track of an ongoing approximation to the Fourier coefficients as follows. A cascade element is called an *arrow-structure* and is defined to consist of the specification of

- an arithmetic arrow A of generation g,
- the wavelet coefficient e_A on $\tilde{\vartheta}_A$ in the approximation of f,
- the coefficients α, β, γ of a trigonometric function defined as follows. The endpoints of the chord corresponding to A decompose C into two circular arcs, exactly one of which contains none of the points $-1, +1, -i$. The sum $\sum_{B \neq A} e_B \tilde{\vartheta}_B$ over all arithmetic arrows B of generation at most g except for A itself is given on this interval by the particular trigonometric function $\alpha \cos\theta + \beta \sin\theta + \gamma$.

The coefficients α, β, γ in an arrow-structure keep a lagged/updated running tally of the overall effect of what has come before it in the cascade; as a result of this technique, the method requires essentially no memory other than the storage of the running approximation to Fourier coefficients and the stack required for the recursive computation in the cascade of arrow-structures.

The next technical point involves a regularization in the calculation of arithmetic wavelet coefficients; this regularization is required because the approximation to f as a linear combination $\sum_{A \in S} e_A \tilde{\vartheta}_A$ is not uniquely determined. Given an arrow-structure in the specified notation and referring to Figure 4, when sampling the input data at the point $\frac{(b+d)+i(a+c)}{(b+d)-i(a+c)}$ in the Farey enumeration with corresponding angular coordinate θ_0, the one real input datum $f(\theta_0)$ must determine the two coefficients e_{UA} and e_{TA}. According to the formulas presented in Figures 3a-3e, these new coefficients are constrained by

$$f(\theta_0) = \alpha \left[\frac{(b+d)^2 - (a+c)^2}{(b+d)^2 + (a+c)^2}\right] + \beta \left[\frac{2(a+c)(b+d)}{(b+d)^2 + (a+c)^2}\right] + \gamma + \frac{4(e_{UA} - e_{TA})}{(b+d)^2 + (a+c)^2} + \frac{8\, e_A \,\mathrm{sgn}\,(a-c)}{(b+d)^2 + (a+c)^2},$$

where sgn $(a-c)$ is a sign defined to be $+1$ if $a > c$ and -1 otherwise. There remains, however, one real degree of freedom in the specification of e_{UA}, e_{TA}, and it is eliminated by demanding the best least-squares fit to the values of f at the next generation of points $\frac{(2b+d)+i(2a+c)}{(2b+d)-i(2a+c)}, \frac{(b+2d)+i(a+2c)}{(b+2d)-i(a+2c)}$ with respective angular coordinates θ_1, θ_2, as illustrated in Figure 4. Updating the lagged trigonometric coefficients in accordance with the formulas presented in Figures 3a-3e produces trigonometric functions $V_1(\theta)$ and $V_2(\theta)$ which give the values of the ongoing approximations at θ_1 and θ_2, respectively, and

whose coefficients are e icit inhomogeneous linear functions of e_l e_{TA}, namely,

$$V_i(\theta_i) = \text{vic} + \text{viu}\, e_{UA} + \text{vit}\, e_{TA}, \quad \text{for } i = 1, 2.$$

The coefficients are explicitly described most concisely in pseudo-code, where e = e_A, alp = α, bet = β, gam = γ, bpd = $b + d$ and apc = $a + c$:

```
if(a > c){
    ax1=alp+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e;
    ay1=2*(b*b-a*a);
    bx1=bet+(c*d-a*d-b*c-a*b)*4*e;
    by1=4*a*b;
    cx1=gam+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e;
    cy1=-2*(a*a+b*b);
    ax2=alp+4*(d*d-c*c)*e;
    ay2=2*(c*c-d*d);
    bx2=bet+8*e*c*d;
    by2=-4*c*d;
    cx2=gam-4*e*(c*c+d*d);
    cy2=2*(c*c+d*d);
    }
else{
    ax1=alp+4*e*(a*a-b*b);
    ay1=2*(b*b-a*a);
    bx1=bet-8*e*a*b;
    by1=4*a*b;
    cx1=gam+4*e*(a*a+b*b);
    cy1=-2*(a*a+b*b);
    ax2=alp+(a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e;
    ay2=2*(c*c-d*d);
    bx2=bet+(a*d-a*b+b*c+c*d)*4*e;
    by2=-4*c*d;
    cx2=gam+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e;
    cy2=2*(c*c+d*d);
    }
v1c=cx1+ (((b+bpd)*(b+bpd)-(a+apc)*(a+apc))*ax1
    +2*(b+bpd)*(a+apc)*bx1)/((b+bpd)*(b+bpd)+(a+apc)*(a+apc));
v1t=cy1+(((b+bpd)*(b+bpd)-(a+apc)*(a+apc))*ay1
    +2*by1*(b+bpd)*(a+apc))/((b+bpd)*(b+bpd)+(a+apc)*(a+apc));
v1u=8./((b+bpd)*(b+bpd)+(a+apc)*(a+apc));
v2c=cx2+(((d+bpd)*(d+bpd)-(c+apc)*(c+apc))*ax2
    +2*(d+bpd)*(c+apc)*bx2)/((d+bpd)*(d+bpd)+(c+apc)*(c+apc));
v2u=cy2+(((d+bpd)*(d+bpd)-(c+apc)*(c+apc))*ay2
    +2*(d+bpd)*(c+apc)*by2)/((d+bpd)*(d+bpd)+(c+apc)*(c+apc));
v2t=-8./((d+bpd)*(d+bpd)+(c+apc)*(c+apc));
```

It is easy to minimize $\left[f(\theta_1) - V_1(\theta_1)\right]^2 + \left[f(\theta_2) - V_2(\theta_2)\right]^2$ as a function of e_{UA}, e_{TA} subject to the given constraint, and this procedure provides a suitable regularization.

Weighted least-squares .s or other statistical treatments using i. it data values at further generations allow alternative regularizations.

Two other simple regularizations which have also been useful are: to require that $e_{UA} = 0$ if $c \geq a$ and $e_{TA} = 0$ if $a > c$, or, to require that $e_{UA}/e_{TA} = -[a(a+c)+b(b+d)]/[c(a+c)+d(b+d)]$; the latter regularization is related to imposing differentiability of the approximation at θ_0.

The final technical point involves the stopping criteria and terminal processing for the cascade of arrow-structures. The basic stopping parameter is NVAN, where a branch of the cascade terminates whenever there have been NVAN consecutive generations of offspring whose contributions to all coefficients c_n in the bandwidth N have been of magnitude at most EPS. There are further parameters governing stopping criteria which may be introduced depending upon the scale and resolution of features in the input data, including a minimum or maximum number of generations, the size of the angle subtended by the chord of a terminal arrow-structure, and so on.

Another useful stopping criterion is to require that $|e_A|\_||\tilde{\vartheta}_A||$ be negligible for several generations, where $||\tilde{\vartheta}_A||$ denotes the L^2-norm of $\tilde{\vartheta}_A$ for which there is the *a priori* estimate $||\tilde{\vartheta}_A|| \leq 10|ac + bd|^{-5/4}$; that is, one terminates the cascade when NVAN consecutive generations of coefficients e_A have satisfied the condition that $|e_A| < \frac{1}{10}$ EPS $|ac + bd|^{5/4}$.

When NVAN generations of offspring contribute negligibly to the specified bandwidth, the cascade is terminated with those offspring of greatest generation. Suppose that A labels the arrow of such a terminal offspring, adopt the notation in Figure 4, and let $J \subseteq C$ denote the circular arc with endpoints $\frac{b+ia}{b-ia}$, $\frac{d+ic}{d-ic}$ which contains none of the points $-1, -i, +1 \in C$. The final update $\tau_i =$ alp$i$ cos $\theta +$ bet$i$ sin $\theta +$ gam$i$, for $i = 1, 2$, of the running trigonometric function on J in accordance with Figures 3a-3e is prescribed in pseudo-code with notation as before as follows:

```
if(a > c){
    alp1=alp+e*2*(a*(a+c)+d*(d-b)-b*(b+d)-c*(c-a));
    bet1=bet+e*4*(c*(d-b)-a*(d+b));
    gam1=gam+e*2*(a*(a+c)+c*(a-c)+b*(b+d)-d*(d-b));
    alp2=alp+e*4*(d-c)*(d+c);
    bet2=bet+e*8*c*d;
    gam2=gam-e*4*(c*c+d*d);
    }
else{
    alp1=alp+e*4*(a-b)*(a+b);
    bet1=bet-e*8*a*b;
    gam1=gam+e*4*(a*a+b*b);
    alp2=alp+e*2*(a*(a-c)+b*(d-b)-c*(c+a)+d*(b+d));
    bet2=bet+e*4*(a*(d-b)+c*(b+d));
    gam2=gam+e*2*(a*(a-c)-c*(a+c)-b*(d-b)-d*(b+d));
    }
```

Now serially letting τ denote the trigonometric functions τ_1 and τ_2, again adopt the notation of Figure 4, where A labels the arrow corresponding to τ. The updated τ is compared with another trigonometric function σ defined by extrapolation which is

uniquely determined by demanding that it agree with f at the three angles comprising the next two generations, $\theta = \theta_0, \theta_1, \theta_2$ in the notation of Figure 4. The Fourier coefficients of the piecewise trigonometric function which takes values $\sigma(\theta) - \tau(\theta)$ on the circular arc J and value zero otherwise are easily computed using standard formulas and added to those of the ongoing approximation.

Other algebraic or statistical samplings of higher generation offspring allow alternative final processings.

For instance, one might save computational expense and simply terminate the cascade with no final processing at all.

Furthermore, the terminal arrow-structures could be stored for restart or iterative refinement capabilities.

One might alternatively record in an arrow-structure different transforms of the trigonometric function defined by α, β, γ. For instance, it is useful to take advantage of the renormalization property of wavelets and change these coefficients by transforming the vector field $[\alpha \cos\theta + \beta \sin\theta + \gamma]\frac{\partial}{\partial\theta}$ by the change of coordinates M_A^{-1}. This technique avoids the necessity of calculating certain large integers attendant to the calculation of wavelet coefficients as will be illustrated later.

One favorable aspect of the method is its advantageous mix of floating-point and integer operation types. Moreover, except for the coefficients $c_0, c_{\pm 1}$, the implementation of the algorithm is purely algebraic, that is, requires only addition and multiplication. Furthermore, by its very nature as a cascade, the algorithm is amenable to parallelization and efficient hardware implementation.

This completes the description of the implementation of the method of calculating Fourier coefficients using arithmetic wavelets.

Several points will next be addressed which are special to the implementation of the methods using modular wavelets. Again there is a basic Step 1, the calculation of the modular wavelet transform where the input function is approximated as $f(\theta) \approx \sum_A g_A \psi_A(\theta)$, and a basic Step 2, substitution of known expressions for the Fourier coefficients of the modular wavelets; these two steps are again conveniently merged into a binary cascade of arrow-structures in practice.

To elucidate Step 1 for modular wavelets, refer again to Figure 4 for the notation near the arithmetic arrow labeled by the matrix A. The single value of the input function $f(\theta_0)$ together with the ongoing calculation of the updated trigonometric function $\alpha \cos\theta + \beta \sin\theta + \gamma$ in the arrow-structure this time uniquely determines the modular wavelet coefficient

$$g_A = \frac{f(\theta_0) - (\alpha \cos\theta_0 + \beta \sin\theta_0 + \gamma)}{\psi_A(\theta_0)}$$

since $\psi_B(\theta_0) = 0$ for every labeling matrix B of generation greater than A; there is thus no need for regularization with modular wavelets.

As to Step 2, recall that a basis of modular wavelets was given before by a collection $\psi_B(\theta)$ of wavelets where A is the label on an arithmetic arrow and either $B = A$ or

$B = SA$. In each case, we can calculate that $\psi_B(\theta)$ has Fourier expansion

$$\psi_B(\theta) \sim d_0^B + d_1^B e^{i\theta} + d_{-1}^B e^{-i\theta} + \frac{i}{2\pi} \sum_{|n|>1} \frac{e^{in\theta}}{n^3 - n} [x_n^B + n\, y_n^B + n^2\, z_n^B],$$

where

$$x_n^B = \begin{cases} 2\,(c^2+d^2)\left[\left(\frac{d-ic}{d+ic}\right)^n - \left(\frac{b-ia}{b+ia}\right)^n\right]; & \text{if } B = A, \\ 2\,(a^2+b^2)\left[\left(\frac{b-ia}{b+ia}\right)^n - \left(\frac{d-ic}{d+ic}\right)^n\right]; & \text{if } B = SA, \end{cases}$$

$$y_n^B = \begin{cases} -4i\,\frac{(ac+bd)}{a^2+b^2}\,\left(\frac{b-ia}{b+ia}\right)^n; & \text{if } B = A, \\ 4i\,\frac{(ac+bd)}{c^2+d^2}\,\left(\frac{d-ic}{d+ic}\right)^n; & \text{if } B = SA, \end{cases}$$

$$z_n^B = \begin{cases} \frac{4}{a^2+b^2}\,\left(\frac{b-ia}{b+ia}\right)^n; & \text{if } B = A, \\ \frac{4}{c^2+d^2}\,\left(\frac{d-ic}{d+ic}\right)^n; & \text{if } B = SA, \end{cases}$$

and

$$2\pi d_0^B = \begin{cases} 4\,\frac{ac+bd}{a^2+b^2} + 2\,(c^2+d^2)[\tan^{-1}\frac{2ab}{b^2-a^2} - \tan^{-1}\frac{2cd}{d^2-c^2}]; & \text{if } B = A, \\ -4\,\frac{ac+bd}{c^2+d^2} - 2\,(a^2+b^2)[\tan^{-1}\frac{2ab}{b^2-a^2} - \tan^{-1}\frac{2cd}{d^2-c^2}]; & \text{if } B = SA, \end{cases}$$

$$2\pi \bar{d}_{-1}^B = 2\pi d_1^B = \begin{cases} -2\,\frac{ac+bd+2i}{(b+ia)^2} + (c+id)^2[\tan^{-1}\frac{2ab}{b^2-a^2} - \tan^{-1}\frac{2cd}{d^2-c^2}]; & \text{if } B = A, \\ 2\,\frac{ac+bd-2i}{(d+ic)^2} - (a+ib)^2[\tan^{-1}\frac{2ab}{b^2-a^2} - \tan^{-1}\frac{2cd}{d^2-c^2}]; & \text{if } B = SA. \end{cases}$$

Thus, the method of calculating Fourier transforms using modular wavelets is easily derived from the method using arithmetic wavelets. In fact, it is a simple matter to incorporate the two minor modifications just described and alter the included source code employing arithmetic wavelets to produce source code employing modular wavelets.

In the same way for fan wavelets, there is again a Step 1 and Step 2, which are merged into a binary cascade of arrow-structures. As for arithmetic wavelets, Step 1 for fan wavelets again requires a regularization scheme. As to Step 2, the formulas for Fourier coefficients of fan wavelets can be derived from those given before for modular wavelets using the identity $\phi_A(\theta) = \psi_A(\theta) - \psi_{UA}(\theta)$ mentioned before. Again, the included source code is easily modified to employ fan wavelets.

Flow Charts for the Fourier Transform

Flow charts for various procedures used in the calculation of Fourier transforms are presented in Figures 5-8. For definiteness, the flow charts and source code refer to arithmetic wavelets without renormalization, but the renormalizing source code will also be included for completeness. For purposes of brevity in these flow charts, the arrow-structures defined before are referred to in these figures simply as "arrows".

In Figure 5 is presented a flow chart for the main driving routine. The input data is normalized as indicated in program segment **1**. There are separate recursions established in program segments **2** and **3** for the top and bottom of the circle respectively.

Each recursion is estab. .ed with a call to the subroutine gener, w h then calls itself in turn. The Fourier coefficients are stored internally with an overall suppressed factor of π, and output data normalization of multiplying by π is accomplished in program segment 4. Output Fourier coefficients are finally displayed before exiting in program segment 5. Program segments 2 and 3 are entirely independent and could be performed in parallel; more generally, each of program segments 2 and 3 could be decomposed further into multiple parallel procedures.

A flow chart for the main recursive routine gener is given in Figure 6. The procedure starts with a test in program segment 6 to determine if:

- the argument envy >= NVAN, in which case there have been NVAN consecutive generations of negligible contributions, and
- the argument generation >= MING, in which case there have been a required minimum number of iterations in the recursion.

If both of these inequalities hold, then the procedure passes to program segment 15, where the cascade is terminated and the output data corrected with a call to the subroutine prune.

In the contrary case that the recursion should continue, the procedure passes to program segment 7, where the descendant arrows in the cascade are determined using the least-squares fit to the next generation of data as described before to compute the regularized wavelet coefficients of the descendants. These are combined with integer calculations to update the lagged trigonometric functions. The procedure then passes to program segment 8, where the ongoing approximations to Fourier coefficients are updated to include contributions from the two descendant arrows with a call of the subroutine charles for each descendant. The procedure continues with a test in program segment 9. If the contribution calculated in the subroutine charles for either descendant to any Fourier coefficient in the bandwidth was non-negligible, then the recursive argument envy is set to zero in program segment 10. In the contrary case that both contributions to all Fourier coefficients in the bandwidth were negligible, the control parameter envy is increased by one in program segment 13.

In any case, the procedure passes to program segment 11, where there is a test on the number of generations. In case too many iterations of the recursion have occurred, it is terminated in program segment 12, and suitable warning of non-convergence of the method is written to the output. In the contrary case that the maximum number of generations has not been reached, program segment 14 establishes the recursion by calling gener once for each of the descendant arrows with the updated control parameter envy and an incremented generation.

In Figure 7 is presented a flow chart for the subroutine charles. Calling the subroutine charles has the effect of updating the ongoing approximations to the Fourier coefficients for a single argument arrow and returning a flag which keeps track of whether any such contribution has been non-negligible. The flag is cleared in program segment 16, and several preliminary calculations are accomplished in program segment 17. The procedure passes to program segment 18, where it is determined whether to calculate the 0,+1,-1 Fourier coefficients. If these are to be calculated, then the procedure passes to program segment 19, which calls a subroutine to update these three Fourier coefficients. Program segments 20 and 21 set the flag as required depending upon whether these three contributions are non-negligible.

In any case, the proce re passes to program segment **22**. If the op over the bandwidth is complete, then all Fourier coefficients have been updated, so the subroutine charles returns the flag in program segment **23**. In the contrary case, the current Fourier coefficient is updated in program segment **24**. The flag is set as required depending on whether the current contribution was non-negligible in program segments **25** and **26**. The iteration over bandwidth is established by the update in program segment **27** and the return to program segment **22**.

In Figure 8 is presented a flow chart for the subroutine prune, which modifies the array of Fourier coefficients when terminating the cascade. The final update of lagged trigonometric functions to produce τ_1, τ_2 is performed in program segment **28**. There is one such function for each possible descendant of the argument arrow. The procedure passes to program segment **29**, where further input data is sampled in order to compute two trigonometric extrapolations σ_1, σ_2, one such extrapolation for each possible descendant of the argument arrow. Each function $\sigma_i - \tau_i$, for $i = 1, 2$, is truncated as described before, and the Fourier coefficients of the truncated functions are added to the ongoing approximations of Fourier coefficients in program segment **30**. Program segments **31** and **32** implement the calculation of 0,+1,-1 Fourier coefficients if desired, and the subroutine prune is terminated with the return in program segment **33**.

The Inverse Fourier Transform

Included in the section entitled "Source Codes" is an implementation in the computer language C of a preferred embodiment of the method described in this section employing arithmetic wavelets.

Since the method for calculating the inverse Fourier transform is similar in spirit and execution to that for the Fourier transform described in detail before, it need only be briefly discussed. The input data includes the specification of a collection of Fourier coefficients c_n in a given bandwidth N.

There are again two main steps in the calculation:

> Step1 : Calculate arithmetic wavelet coefficients e_A from the given Fourier coefficients c_n.
>
> Step 2: Use the wavelet coefficients from Step 1 and the reconstruction algorithm to output values of the function $\sum_A e_A \tilde{\vartheta}_A(\theta)$ at the Farey quadrature points on the circle.

For Step 1, there is again an explicit formula from previous theoretical work. Deep mathematical results prove that for $n \neq 0, \pm 1$,

$$e^{in\theta} = \cos n\theta + i \sin n\theta$$
$$= -\left[c_0^n + c_1^n e^{i\theta} + c_{-1}^n e^{-i\theta}\right] + \frac{i}{4} \sum_A \left\{ n(\xi^n + \eta^n) + \frac{\eta + \xi}{\eta - \xi} (\xi^n - \eta^n) \right\} \overline{\vartheta}_A(\theta),$$

where the sum is over all arithmetic arrows, the underlying chord of which has endpoints

$\xi, \eta \in C$, and

$$c_0^n = \begin{cases} -1, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -1, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_1^n = \begin{cases} 0, & n \equiv 0(4); \\ -1, & n \equiv 1(4); \\ i, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_{-1}^n = \begin{cases} 0, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -i, & n \equiv 2(4); \\ -1, & n \equiv 3(4). \end{cases}$$

Substituting this expression for $e^{in\theta}$ into the given Fourier expansion $f(\theta) \sim \sum_n c_n e^{in\theta}$ gives

$$f(\theta) \approx c_0' + c_1' e^{i\theta} + c_{-1}' e^{-i\theta} + \sum_A e_A \, \overline{\vartheta}_A(\theta),$$

where

$$e_A = \frac{i}{4} \sum_{n \neq 0, \pm 1} c_n \left\{ n(\xi^n + \eta^n) + \frac{\eta + \xi}{\eta - \xi} \, (\xi^n - \eta^n) \right\},$$

and

$$c_\ell' = c_\ell - \sum_{n \neq 0, \pm 1} c_n \, c_\ell^n, \quad \text{for } \ell = 0, \pm 1.$$

This expression $f(\theta) \approx c_0' + c_1' e^{i\theta} + c_{-1}' e^{-i\theta} + \sum_A e_A \, \overline{\vartheta}_A(\theta)$ of $f(\theta)$ constitutes Step 1 in the calculation of inverse Fourier transform.

In fact, the calculation of wavelet coefficients e_A in Step 1 can be implemented using mostly integer operations to improve run-time dramatically. To see this, notice that the formulas $\xi = \frac{b+ia}{b-ia}$, $\eta = \frac{d+ic}{d-ic}$ for the endpoints of the chord underlying the arrow labeled by the matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ may be substituted into the expression for e_A to yield

$$e_A = \frac{i}{4} \sum_{n \neq 0, \pm 1} c_n \, e_A^n,$$

where

$$e_A^n = \frac{[(bd - ac) + i(ad + bc)]^n}{(a^2 + b^2)^n (c^2 + d^2)^n} \times$$

$$\left[n\{(bd + ac + i)^n + (bd + ac - i)^n\} + i(bd + ac)\{(bd + ac + i)^n - (bd + ac - i)^n\} \right].$$

In the obvious implementation of this formula to calculate e_A^n using integer operations, the intermediary integer expressions grow too large to be practical for n large if the arrow corresponding to A is of large generation.

This difficulty is overcome by wavelet renormalization, where these attendant large integer calculations are obviated entirely as is illustrated in the included source code.

Step 2 in the method for calculating inverse Fourier transforms depends upon the reconstruction algorithm to output function values on the circle taken by the linear combination $f(\theta) \approx c_0' + c_1' e^{i\theta} + c_{-1}' e^{-i\theta} + \sum_A e_A \, \overline{\vartheta}_A(\theta)$. The finiteness property of the reconstruction algorithm allows the exact calculation of these function values at the sample points $Q \subseteq C$ in their ordering determined by the Farey quadrature. The two steps are again conveniently merged into a single binary cascade as follows.

There is only one control parameter, which is called SCALE, where the method is required to determine at least one output value in each circular arc subtending an angle SCALE. Thus, as SCALE is decreased the grid of output values on the circle is refined. A binary cascade is established using arrow-structures which is similar to that for the Fourier transform before. However, the wavelet coefficients are given in this case in closed form by the formula for e_A (so no least-squares fit is required). Because of the finiteness property of the reconstruction algorithm, exact output values are determined at each stage of the recursion. The cascade is terminated with an arrow whenever the chords underlying both descendant arrows subtend angles less than SCALE. Output data can be written to an array either when calculated to store data in the ordering of the Farey quadrature or when terminating the cascade to store data in the standard ordering on the circle. Furthermore, the terminal arrow-structures could be stored for restart or iterative refinement capabilities.

This completes the description of the implementation of the method of calculating the inverse Fourier transform using arithmetic wavelets.

Several points will next be addressed which allow the extension of the methods to modular and fan wavelets. Again there is a basic Step 1, the calculation of wavelet coefficients from Fourier coefficients, and a Step 2, the reconstruction algorithm; in each case of fan or modular wavelets, these two steps are again conveniently merged into a binary cascade of arrow-structures in practice. The finiteness condition on fan or modular wavelets mentioned before renders Step 2 entirely analogous to that for arithmetic wavelets. As to Step 1, simply substitute the identities $\overline{\vartheta}_A = \phi_A - \phi_{UA} = \psi_{UA} - 2\psi_A + \psi_{U^{-1}A}$ described before into the expression above for $e^{in\theta}$ in terms of arithmetic wavelets to immediately derive corresponding expressions in terms of fan and modular wavelets.

Thus, the method of calculating inverse Fourier transforms using fan or modular wavelets is easily derived from the method using arithmetic wavelets. In fact, it is a simple matter to incorporate the minor modifications just described and alter the included source code employing arithmetic wavelets to produce source code employing fan or modular wavelets.

Flow Charts for the Inverse Fourier Transform

Flow charts for various procedures used in the calculation of inverse Fourier transforms are presented in Figures 9-10. For definiteness, the flow charts and source code refer to arithmetic wavelets without renormalization, but the re-normalizing source code will also be included for completeness. For purposes of brevity in these flow charts, the arrow-structures defined before are referred to in these figures simply as "arrows".

In Figure 9 is given a flow chart for the main driving routine. Modified Fourier coefficients c'_0, c'_1, c'_{-1} are computed in program segment **34**. The formula given in Step 1 for the inverse Fourier transform is applied in program segment **35** to calculate the arithmetic wavelet coefficients in terms of Fourier coefficients for a particular family of nine arrows. These nine arithmetic wavelet coefficients are required to initialize the trigonometric functions for recursions established in program segment **36** with calls to the recursive subroutine gener. Recursions with initial conditions corresponding to $A = U, T, T^{-2}, U^{-1}, T^{-1}, T^{-1}U^{-1}, U^{-2}$ are undertaken in program segment **36**. These

recursions, as well as fi ıer possible subrecursions derived from m, are independent and may be undertaken in parallel. This elaborate driving routine is convenient because $e^{in\theta}$ is expressed in Step 1 of the inverse Fourier transform method in terms of the *normalized* arithmetic wavelets $\overline{\vartheta}_A$ which differ from the arithmetic wavelets $\tilde{\vartheta}_A$ themselves only for $A = U^{-1}, T^{-1}$.

The procedure passes to program segment **37** to perform an overall correction of the function values using the modified Fourier coefficients c'_0, c'_1, c'_{-1}. Output data is displayed and the procedure terminated in program segment **38**.

In Figure 10 is given a flow chart for the main recursive routine gener.. In the cascade, the two descendant arrows of the argument arrow are generated in program segment **39** employing Step 1 of the inverse Fourier transform method to calculate the wavelet coefficients of the descendants. These expressions are then used to update the lagged trigonometric functions in program segment **39**. A test is performed in program segment **40** to determine if the chord underlying each descendant arrow subtends a sufficiently small angle, as estimated by the comparison of an integer quantity with SCAT; the relation with the control parameter SCALE discussed before is given by SCAT = $[\exp \sinh^{-1}(1/\text{SCALE})]^2$. In case one of the chords subtends too large an angle, the procedure passes to program segment **41**. The recursion is established in program segment **41** with two calls to gener, where the arguments are given by the two current descendant arrows. In the contrary case, the procedure performs the final update of the lagged trigonometric functions in program segment **42**, then stuffs the output array with the new function values in program segment **43**, and finally returns in program segment **44**.

Data Compression

As with any method for data compression based on transform coding, five basic manipulation are required, namely:

- wavelet filtering,
- quantization,
- storage and retrieval or transmission
- de-quantization, and
- reconstruction.

The wavelet filter has already been fully disclosed as Step 1 for the calculation of the Fourier transform, and the wavelet inverse filter or reconstruction algorithm has likewise already been described as Step 2 for the calculation of the inverse Fourier transform. Application-specific quantization is done according to psychovisual or psychoacoustic thresholds. Quantization and storage are furthermore merged with wavelet filtering into a single binary cascade as before, and retrieval or transmission are likewise merged with reconstruction into another single binary cascade. The reconstruction algorithm is exact. Furthermore, source code for it may be transmitted along with compressed data since the coding is sufficiently abbreviated and low-level.

As is well-known, useful compression by transform coding requires an efficient specifi-

cation of the basis eler. .ts. This is especially advantageous for ! new wavelets: a top arrow of generation g labeled by the two-by-two integral matrix A is specified by g bits, namely, by writing A uniquely as the matrix product of factors U or T. For example, the matrix $A = \begin{pmatrix} 13 & 9 \\ 10 & 7 \end{pmatrix}$ factors as

$$\begin{pmatrix} 13 & 9 \\ 10 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

so the coefficient for the wavelet $\tilde{\vartheta}_A$ is coded by the binary sequence $TUUUTTU$.

The method is also especially well-suited to progressive picture build-up or other iterative refinement as follows. The Farey quadrature determines a linear ordering on the set Q of quadrature points in the circle C as before. The ordered subset $Q\backslash\{-i, \pm 1\} \subseteq Q \subseteq C$ is put into bijective correspondence with the collection of all arithmetic arrows (where $\backslash$ denotes the set-theoretic relative complement); as illustrated in Figures 1a-1c, the arithmetic arrow labeled by $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ corresponds to the complex number

$$\begin{cases} \frac{(b+d)+i(a+c)}{(b+d)-i(a+c)} \in C, & \text{for a top arrow;} \\ \frac{(b-d)+i(a-c)}{(b-d)-i(a-c)} \in C, & \text{for a bottom arrow;} \\ i \in C, & \text{for the doe.} \end{cases}$$

Thus, the ordering from the Farey quadrature on $Q\backslash\{-i, \pm 1\}$ determines a corresponding ordering on the set of all arithmetic wavelets. Storage, retrieval, transmission, and reconstruction of wavelet coefficients is accomplished at a specified input or output data resolution in this canonical ordering since energy compacts to the lesser wavelet coefficients. In the case of real-time compression and transmission or other manipulation not requiring retrieval, the method can be further improved by the manipulation of previously computed wavelet coefficients in parallel with the calculation of subsequent ones.

Complex and Multi-Dimensional Data

There is nothing remarkable about the extension of the invention to complex-valued input data $f(\theta) = u(\theta) + i\, v(\theta)$, where u and v are real-valued functions on the circle C. Concentrating for definiteness on arithmetic wavelets, one simple approach is to separately and independently transform u and v,

$$u(\theta) \approx \sum e_A\, \tilde{\vartheta}_A(\theta) \text{ and } v(\theta) \approx \sum f_A\, \tilde{\vartheta}_A(\theta),$$

using the techniques already described in order to calculate

$$f(\theta) \approx \sum (e_A + i f_A)\, \tilde{\vartheta}_A(\theta).$$

Another direct approach is to allow complex values for the input function f and solve for complex wavelet coefficients using the same algebraic formulas as before.

Turning now to multi-dimensional input/output data, suppose that there are $R \geq 1$ input functions each of which has $M \geq 1$ independent variables, and define the *torus* C^M to be the M-fold Cartesian product

$$C^M = \underbrace{C \times \cdots \times C}_{M \text{ times}}.$$

Coordinates on C^M are given by M-tuples $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_M)$ of angles, and the input data is specified by an R-tuple of multivariable real- or complex-valued functions

$$F(\vec{\theta}) = (f^1(\vec{\theta}), f^2(\vec{\theta}), \ldots, f^R(\vec{\theta})).$$

Using the bijection described in the previous section on data compression, each quadrature point $Z \in Q\backslash\{-i, \pm 1\}$ uniquely determines a corresponding arithmetic wavelet $\tilde{\vartheta}_Z(\theta)$; by convention, one also defines $\tilde{\vartheta}_{-i}(\theta) = \tilde{\vartheta}_{\pm 1}(\theta) = 1$. Concentrating on arithmetic wavelets for definiteness, define the corresponding *multiwavelet*

$$\tilde{\vartheta}_{\vec{Z}}(\vec{\theta}) = \tilde{\vartheta}_{Z_1}(\theta_1)\ \tilde{\vartheta}_{Z_2}(\theta_2)\ \ldots \tilde{\vartheta}_{Z_M}(\theta_M).$$

As before, the input function F must be normalized. To this end, define a *trigonometric monomial* to be a product of the form $X^1(\theta_1)X^2(\theta_2)\cdots X^M(\theta_M)$, where each $X^i(\theta)$ is given by one of $X^i(\theta) = \cos\theta$, $X^i(\theta) = \sin\theta$, or $X^i(\theta) = 1$. The values of F at the points in $\{-i, \pm 1\}^M$ uniquely determine coefficients in a linear combination $\nu(\vec{\theta})$ of trigonometric monomials so that $\nu(\vec{\theta}) = F(\vec{\theta})$ for each point of $\{-i, \pm 1\}^M$. Just as in the one-dimensional wavelet filter already discussed, now one must consider the *normalization* $\overline{F}(\vec{\theta}) = F(\vec{\theta}) - \nu(\vec{\theta})$, so $\overline{F}$ is zero at each point of $\{-i, \pm 1\}^M$.

Adopting the Farey quadrature in each factor circle C of the torus C^M, there is an induced lexicographic ordering on the set $Q^M\backslash\{-i, \pm 1\}^M \subseteq C^M$ of potential sample points $\vec{Z} \in C^M$. One separately approximates each coordinate function

$$f^i(\vec{\theta}) \approx \sum e^i_{\vec{Z}}\ \tilde{\vartheta}_{\vec{Z}}(\vec{\theta}),\ \text{for } i = 1, 2, \ldots, R,$$

of $\overline{F}(\vec{\theta})$ as a finite linear combination of multiwavelets with application-specific zonal sampling for optimum energy compaction; this calculation may employ least-squares or other regularization in each factor as well as renormalization as for the one-dimensional wavelet filter. The methods are again elegantly implemented in practice as M nested binary cascades. The final processing for the cascades of arrow-structures in the calculation of multi-dimensional Fourier transforms may involve adding suitable linear combinations of trigonometric monomials based on further sampling, or one may simply truncate with no final processing at all, as in the one-dimensional case.

These same remarks apply verbatim to fan and modular wavelets.

Mathematical Basis

"The decorated Teichmüller space of punctured surfaces", *Communications in Mathematical Physics* **13**, pp. 299-339 (1987), written by the author of this patent application, began the study of certain abstract geometric coordinates called *lambda lengths*

in the context of hype olic geometry on Riemann surfaces. Th leas developed in this and other related papers have found applications in high energy physics. A more recent publication in this series of about 20 papers is entitled "Universal constructions in Teichmüller Theory", *Advances in Mathematics* **98**, pp. 143-215 (1993). Here it was shown how to generalize lambda lengths to provide coordinates on a certain space of homeomorphisms (i.e. continuous bijections whose inverses are also continuous) of the circle, where there is one generalized lambda length for each arithmetic arrow. In further recent work described in "The Lie algebra of homeomorphisms of the circle", *Advances in Mathematics* **140**, pp. 282-322 (1998), written jointly with Feodor Malikov, there is described a representation of the coordinate deformations of these generalized lambda lengths as explicit vector fields on the circle. These vector fields are called "normalized elementary vector fields", and their study led to other vector fields on the circle called "fans" and "hyperfans". These three families of vector fields correspond, respectively, to the arithmetic, fan, and modular wavelets described here via the identification of the vector field $f(\theta)\frac{\partial}{\partial\theta}$ with the function $f(\theta)$; there was, however, no discussion of wavelets, sampling of data, approximation of functions, or any algorithms in the published literature.

Each arithmetic wavelet is once-continuously differentiable on the circle, compactly supported, and localized in space. Arithmetic wavelets are not compactly supported in frequency, but instead, the frequency profile of an arithmetic wavelet is given algebraically by the formula for c_n^A used in Step 2 in the calculation of Fourier transforms; a non-compactly supported localization in frequency follows directly from this. The asserted formula for c_n^A can be derived without much difficulty but in several cases directly from Figures 3a-3e integrating twice by parts the usual expression for Fourier coefficients using the fact ϑ_A is once-continuously differentiable. More difficult to derive is the formula given for the exponential functions $e^{in\theta}$ in terms of arithmetic wavelets which was used in Step 1 of the calculation of inverse Fourier transforms; this subtle computation with lambda lengths is given in "The Lie algebra of homeomorphisms of the circle".

Similar remarks apply to fan and modular wavelets, but fan wavelets are only continuous (they are not differentiable), and modular wavelets are not even continuous.

In order to explain the failure of orthonormality of arithmetic wavelets and illuminate the regularization used in the arithmetic wavelet filter, fix some matrix A labeling an arithmetic arrow whose underlying chord has initial point q in the circle C. The sequence U^nA of matrices label the arithmetic arrows whose underlying chords also have q as initial point, where n denotes an arbitrary integer. There is one infinite linear dependence

$$0 = \sum_n \overline{\vartheta}_{U^nA}.$$

for each $q \in Q$, and this explains the additive degree of freedom in the calculation of e_{UA} and e_{TA} in the wavelet filter which is handled by regularization as was discussed before. This additive degree of freedom is also manifest in the initial specification $e_I = 0$ of wavelet coefficient on the doe in the wavelet filter.

In order to better understand the Farey quadrature, it is convenient to transform the

disk D to the upper h plane $\mathcal{U} = \{u + iv : v > 0\}$ by means of transform

$$K : D \to \mathcal{U}$$
$$z \mapsto i\,\frac{z+1}{z-1}.$$

This function K is a homeomorphism on the interior of D which maps the unit circle C bijectively to the real-axis $\{u + iv : v = 0\}$ together with an additional point ∞ at infinity, where $K(-1) = 0$, $K(-i) = 1$, and $K(+1) = \infty$. Furthermore, as $q \in Q$ varies over all quadrature points, the real number $K(q)$ varies over all rational numbers, where the rational number $\frac{p}{q}$ corresponds to the quadrature point $\frac{p-iq}{p+iq} \in Q$. In order to illustrate the Farey tesselation itself in $\mathcal{U}$, whenever $q_1, q_2 \in Q$ are the endpoints of the chord underlying an arithmetic arrow in D, draw the semi-circle in $\mathcal{U}$ with real endpoints q_1, q_2; in the degenerate case that one of the quadrature points is infinite, say the point $q_2 = \infty$, then draw the vertical ray in $\mathcal{U}$ with endpoint q_1. Applying this procedure to each arithmetic arrow produces the classical model of the Farey tesselation in $\mathcal{U}$ that is indicated in Figure 11. This figure illustrates the relationship between the Farey tesselation and the indicated circle packing, where the circle in the figure that is tangent to the real-axis at the rational point p/q has diameter q^{-2}. For further information on number-theoretic aspects of this classical figure, see, for instance, Ford's book "Automorphic Functions", Chelsea (1972).

Source Codes

In order to adequately disclose the methods, it is the purpose of this paragraph to include source codes for their implementations in the computer language C. Source code is presented for each of the following two implementations.

- awft is the implementation of a preferred embodiment of the method for computing the Fourier transform of real-valued input data defined on the circle. The algorithm depends upon a bandwidth N, tolerance EPS, generation cut-off NVAN, and minimum generation MING as described before. A further flag SMODE=1 determines that the output data will include the 0,+1,-1 Fourier coefficients, and SMODE=0 otherwise. (The only configuration in either code which requires any library routines is SMODE=1, which requires math.h.)

- awift is the implementation of a preferred embodiment of the method for computing the inverse Fourier transform of specified complex-valued Fourier coefficients defined in a specified bandwidth N. The output data is controlled by a single parameter SCAT, which is related to the parameter SCALE discussed before by SCAT=[exp sinh ^ (-1)(1/SCALE)] ^ 2. (SCAT varies inversely with SCALE and may be compared with conveniently derived approximate quantities.)

The comparison of the subroutines tgener in awft and in awift illustrates the easy transition between source codes for real and for complex data. The corresponding complex-awft and real-awift are therefore easily derived from the included source codes.

Furthermore, the met. l for compression of one-dimensional in] data amounts to the first step of awft, then quantization/de-quantization, then the second step of awift. The included source codes together thus also implement the method for compression of one-dimensional data since source code for it may be extracted from the included source codes. Finally, it is straight-forward to write the further driving routine required for multi-dimensional data compression, so the source codes included with this patent application are sufficient to readily implement the method for multi-dimensional data compression as well.

The included implementations have *not* been optimized. (For instance, straight-forward optimization has improved run-time by as much as 70% over the included awft code; on the other hand, optimization destroys intelligibility of the coding in relation to the underlying algorithm, and this explains the inclusion of unoptimized source code.) Another substantial improvement in run-time over the included source codes can be achieved by factoring so as to replace various manipulations of complex numbers with manipulations of integers as discussed before.

Moreover, for clarity, the included source codes do not take advantage of renormalization, but for completeness, to each of awft and awift is appended source code to replace the subroutines tgener and bgener by corresponding subroutines which *do* employ renormalization.

```
//This is an implementation in C of a preferred embodiment
//of the invention for the calculation of the Fourier transform

#include <stdio.h>

#include <math.h>          //math.h is required only for the calculation of 0,+1,-1 Fourier coeffs

#define GEN 1000           //the maximum allowed generation

#define N 50               //output bandwidth

#define COR 1              //irrelevant parameter

#define EPS 1.e-12         //tolerance, where contributions to Fourier coeffs of norm less than
                           //EPS are regarded as negligible

#define PI 3.141592653589793

#define NVAN 1             //terminate the recursion when NVAN consecutive
                           //generations have contributed negligibly

#define MING 15//calculate at least MING generations

#define SMODE 1            //SMODE=1 includes the calculation of 0,+1,-1 Fourier coeffs
                           //SMODE=0 is faster and does not require math.h


struct complex{            //a complex number
        double x;
        double y;
        };

struct trip{               //convenient to have a real triple
        double alp;
        double bet;
        double gam;
        };

struct edge{               //an arrow-structure
        int a,b,c,d;
        double e,alp,bet,gam;
        };

int count=0;               //a running count of the total number of samples of input data

struct complex fourier[2*N+1];   //the complex Fourier coeffs stored in the order
                                 //n=-N,...,-1,0,1,...,N, so fourier(n) is (N+n)th coeff
```

```
double abar,bbar,cbar;          //coeffs in trig function abar*cos theta + bbar*sin theta + cbar
                                //which agrees with input function for theta = -PI, -PI/2, 0

void main(void){

        void genertop(struct edge *,int,int,double);   //recursive routine for top of circle
        void generbot(struct edge *,int,int,double);   //recursive routine for bottom of circle

        void normalout(void);           //output normalization routine
        void normalin(void);            //input normalization routine

        double fbar(int,int); //fbar(p,q) returns f(theta) - (abar*cos theta + bbar*sin theta + cbar)
                              //where theta is the argument of the complex number (p-iq)/(p+iq)

        struct edge doe[1];             //initial edge for recursive calls

        int i;

        normalin();             //this calculates the normalization parameters abar, bbar, cbar

        doe->a=doe->d=1;        //initialize the matrix of the doe to the identity
        doe->b=doe->c=0;
        doe->alp=doe->bet=doe->gam=doe->e=0;    //initialize wavelet and lagged trig coeffs

        genertop(doe,1,0,fbar(-1,1));           //begin recursion on top of the circle

        generbot(doe,1,0,0.);                   //begin recursion on bottom of  the circle

        normalout();            //this divides each output Fourier coeff by an overall factor PI
                                //which was suppressed during the recursion and determines
                                //negative from positive Fourier coeffs using reality of input

        printf("with EPS=%e, NVAN=%d, COR=%d, and MING=%d, the count is: %d\n",
        EPS,NVAN,COR,MING,count);

        for(i=0;i<=N;i++)
                printf("c%3d = %e+(i)%e\n",i,fourier[N+i].x,fourier[N+i].y);

}


void genertop(struct edge *p,int g,int envy,double fc){         //recursion for top of circle

                        //input pointer to a current edge p of generation g, where there
                        //have so far been envy consecutive generations of negligible
                        //contributions, and fc is the current normalized input sample
```

```
struct edge twofer[2];                  // the descendant edges of p

struct complex temp;

void genertop(struct edge *,int,int,double);    //recursive function

struct complex tgener(struct edge *,struct edge *,double);

                        //tgener(p,twofer,f) stuffs twofer with the descendants of p
                        //using the updated normalized input sample f returning the
                        //next two normalized samples as its real and imaginary parts

int charles(struct edge *);  //charles(p) updates the Fourier coeffs with the
                                //contribution from edge p returning 1 if they
                                //are negligible, and returning 0 otherwise


void tprune(struct edge *); //this implements the processing of a terminal edge

if(envy>=NVAN && g>= MING){ //if appropriate, then terminate cascade
        tprune(p);
        return;
        }

temp=tgener(p,twofer,fc);           //otherwise, generate descendants of p

if(charles(twofer)*charles(twofer+1))       //if both descendants contribute negligibly,
        envy++;                             //then increment envy
else
        envy=0;                             //otherwise reset envy to zero

if(g>=GEN){             //problem: non-convergence after GEN generations

        printf("fall through with gen=%d for a,b,c,d=%d,%d,%d,%d and e=%e\n",
                g,p->a,p->b,p->c,p->d,p->e);
        return;
        }

else{                       //otherwise, continue recursion with descendants

        genertop(twofer,g+1,envy,temp.x);
        genertop(twofer+1,g+1,envy,temp.y);
        }

}


void generbot(struct edge *p,int g,int envy,double fc){         //recursion for bottom of circle
```

```
        struct edge twofer[2];                          //this is entirely analogous to genertop

        struct complex temp;

        void generbot(struct edge *,int,int,double);
        struct complex bgener(struct edge *,struct edge *,double);
        int charles(struct edge *);
        void bprune(struct edge *);

        if(envy>=NVAN && g >= MING){
                bprune(p);
                return;
                }

        temp=bgener(p,twofer,fc);

        if(charles(twofer)*charles(twofer+1))
                envy++;
        else
                envy=0;

        if(g>=GEN){
                printf("fall through with gen=%d for a,b,c,d=%d,%d,%d,%d and e=%e\n",
                        g,p->a,p->b,p->c,p->d,p->e);
                return;
                }

        else{
                generbot(twofer,g+1,envy,temp.x);
                generbot(twofer+1,g+1,envy,temp.y);
                }

}


int charles(struct edge * p){           //charles(p) updates the Fourier coeffs using p
                                        //and returns a control parameter flag=1 if
        int n,flag;                     //contribution is negligible and flag=0 otherwise

        struct complex makecpx(int,int);    //input p,q returns the complex number (p-iq)/(p+iq)

        struct complex mult(struct complex,struct complex);  //complex multiplication

        int a,b,c,d;                        //local variables for matrix values
        double e;                           //local variable for wavelet coeff

        struct complex pzl,pzr,pzf,pzb;     //l,r,f,b labels left,right,front,back vertices
                                            //of Farey tesselation near the edge p, and
        struct complex plm,prm,pfm,pbm;     //in a loop below on index n, pzx=(pxm)^n
```

```
                                        //for x=l,r,f,b
struct complex tmp,mul,cn;

int sixupc(int,int,int,int,double);     //this updates the Fourier coeffs 0,+1,-1
                                        //provided SMODE=1


a=p->a;                                 //stuff local values
b=p->b;
c=p->c;
d=p->d;
e=p->e;

pzr.x=b-d;                              //set-up for loop over Fourier coeffs
pzr.y=c-a;
prm=makecpx(b-d,a-c);
pzr=mult(pzr,pzr);


pzl.x=b+d;
pzl.y=-c-a;
plm=makecpx(b+d,a+c);
pzl=mult(pzl,pzl);


pzf.x=d;
pzf.y=-c;
pfm=makecpx(d,c);
pzf=mult(pzf,pzf);


pzb.x=b;
pzb.y=-a;
pbm=makecpx(b,a);
pzb=mult(pzb,pzb);

flag=1;                                 //initialize flag
tmp.x=0;

if(SMODE)
        flag=sixupc(a,b,c,d,e);         //update 0,+1,-1 modes and re-set flag to 0 if any
                                        //of these contributions is non-negligible


for(n=2;n<=N;n++){                      //loop over positive Fourier coeffs n>1

        tmp.y=e/(n-n*n*n);

        pzr=mult(pzr,prm);              //recursively calculate powers
        pzl=mult(pzl,plm);
```

```
                pzf=mult(p. .,pfm);
                pzb=mult(pzb,pbm);


                cn.x=-pzr.x+2*pzf.x+2*pzb.x-pzl.x;
                cn.y=-pzr.y+2*pzf.y+2*pzb.y-pzl.y;

                mul=mult(cn,tmp);

                fourier[n+N].x+=mul.x;    //update nth Fourier coeff
                fourier[n+N].y+=mul.y;

                if(flag==1 && mul.x*mul.x+mul.y*mul.y>=EPS) //if contribution to nth Fourier
                        flag=0;                              //coeff is non-negligible,
                                                             //then set flag to zero
                }

                return(flag);

}


int sixupc(int a,int b,int c,int d,double e){          //this updates the 0,+1,-1 Fourier coeffs
                                                       //using the input matrix entries a,b,c,d
                                                       //and wavelet coeff e

        double tp,tm,tr,tl; //p,m,r,l labels the terminal,initial,right,left vertices of Farey
                            //tesselation near edge, and tx denotes corresponding angle
        double dtemp;

        double tb;          //tb=1 on the top  and tb=-1 on the bottom of the circle

        int fl;             //this is the local version of the flag in charles and is the
                            //value returned to charles by sixupc

        struct complex makecpx(int,int), temp;

        fl=1;               //set the flag to 1

        tb=1.;              //default to the top of the circle
        if(b+c<0)           //if on the bottom of the circle
                tb=-1.;     //then re-set tb=-1.

//calculate the various angles

        temp=makecpx(d,-c);
        tp=acos(tb*temp.x);

        temp=makecpx(b,-a);
```

```
        tm=acos(tb*temp.x),

        temp=makecpx(b+d,-(a+c));
        tl=acos(tb*temp.x);

        temp=makecpx(b-d,c-a);
        tr=acos(tb*temp.x);

        dtemp=e*(                                          //contribution to 0th Fourier coeff
                    2.*tp*(c*c+d*d)+2.*tm*(a*a+b*b)
                    -tl*((a+c)*(a+c)+(b+d)*(b+d))
                    -tr*((a-c)*(a-c)+(b-d)*(b-d))
                    );

        if(dtemp*dtemp>=EPS)    //if this contribution is non-negligible,
                fl=0;           //then re-set the flag

        fourier[N].x+=dtemp;    //up-date the 0th Fourier coeff

        temp.x=e*(                                         //contribution to real part of 1st
                    tp*(c*c-d*d)+tm*(a*a-b*b)              //Fourier coeff
                    +tr*((b-d)*(b-d)-(a-c)*(a-c))/2.
                    +tl*((b+d)*(b+d)-(a+c)*(a+c))/2.
                    );

        fourier[N+1].x+=temp.x;                            //update real part of 1st Fourier coeff



        temp.y=e*(                                         //contribution to imag part of 1st
                    2.*(tp*c*d+tm*a*b)                     //Fourier coeff
                    -tr*(b-d)*(a-c)
                    -tl*(b+d)*(a+c)
                    );

        fourier[N+1].y+=temp.y;                            //update imag part of 1st Fourier coeff

        if(temp.x*temp.x+temp.y*temp.y>=EPS)               //if this contribution is non-negligible,
                fl=0;                                      //then re-set the flag to zero

        return(fl);

}


struct complex mult(struct complex u,struct complex v){    //complex multiplication
        struct complex temp;
```

```
        temp.x=u.x*v.x-u.y    .y;
        temp.y=u.x*v.y+v.x*u.y;

        return(temp);
}

struct complex makecpx(int p,int q){        //input p,q returns the complex number (p+iq)/(p+iq)

        struct complex temp;
        double den;

        den=p*p+q*q;

        temp.x=(p*p-q*q)/den;
        temp.y=-2*p*q/den;

        return(temp);
}

void normalout(void){                       //output normalization

        double fz=0.,fi=0.,fo=0.;
        double alp,bet,gam;
        int n;

        for(n=0;n<=N;n++){
                fourier[N+n].x=fourier[N+n].x/PI;  //include factor PI suppressed before
                fourier[N+n].y=fourier[N+n].y/PI;
                fourier[N-n].x=fourier[N+n].x;     //calculate negative from positve
                fourier[N-n].y=-fourier[N+n].y;    //Fourier coeffs
                }

}

void normalin(void){                        //input normalization

        double fz=0.,fi=0.,fo=0.;           //values of input function for angles -PI, -PI/2, 0

        int i;

        double f(int,int);                  //f(p,q) returns the un-normalized input sample
                                            //at the angle corresponding to the point on the
                                            //circle given by the complex number (p-iq)/(p+iq)
        for(i=0;i<=2*N;i++)
                fourier[i].y=fourier[i].x=0;
```

```
        fz=f(0,1);
        fi=f(1,0);
        fo=f(1,1);

        abar=(fi-fz)/2;    //abar*cos theta + bbar*sin theta + cbar takes the same values
        cbar=(fi+fz)/2;    //as f for the angles theta=-PI, -PI/2, 0
        bbar=cbar-fo;

        fourier[N].x=cbar*PI;          //stuff the 0th Fourier coeff scaled by PI to
        fourier[N].y=0.;               //recover it after re-scaling by 1/PI in normalout

        fourier[N+1].x=abar*PI/2;      //stuff the 1st Fourier coeff scaled by PI to
        fourier[N+1].y=-bbar*PI/2;     //recover it after re-scaling by 1/PI in normalout

}


double fbar(int p, int q){             //input p,q to fbar produces the normalized value
        struct complex temp;           //of the input data at the point (p-iq)/(p+iq)
        struct complex makecpx(int,int);
        double f (int,int);
        temp=makecpx(p,q);
        return(f(p,q) - (abar*temp.x + bbar*temp.y+cbar));
}

double f(int p,int q){                 //f is the input data, where f(p,q) is the
        double fsinm(int,int,int);     //value taken at (p-iq)/(p+iq)
        return(fsinm(p,q,6));
}

double fsinm(int p,int q, int m){      //for example, the input function is
        int n;                         //here taken to be sin(6*theta)
        struct complex zeta,meta;
        struct complex makecpx(int,int),mult(struct complex,struct complex);

        zeta=makecpx(p,q);
        meta.x=1.;
        meta.y=0;

        for(n=1;n<=m;n++)
                meta=mult(zeta,meta);

        return(meta.y);
}


struct complex tgener(struct edge *pc,struct edge *pn,double fc){
                        //tgener(p, twofer,fc) stuffs twofer with the descendants
                        //of p for the top of the circle, where fc is the current
```

```
                //normalized input sample, and tgener returns a complex number.
                //the real and imag parts being the normalized input values
                //required for the least-squares regularization

double fbar(int,int);           //fbar(p,q) returns the normalized input values

int a,b,c,d,bpd,apc;            //local variables for matrix entries, bpd=b+d, apc=a+c

double e,alp,bet,gam;           //local variables for wavelet and lagged trig coeffs

double ax1,ay1,ax2,ay2;         //index 1,2 refers to first,second (counter-clockwise)
double bx1,by1,bx2,by2;         //descendants, prefix a,b,c refers to alp,bet,gam
double cx1,cy1,cx2,cy2;         //update procedures below define indices x,y

double sig1,tau1,sig2,tau2,chi; //quantities used in the least-squares regularization

double eu,et;                   //wavelet coeffs of first and second descendant edges

double fn,fnp;                  //normalized input samples


double v1c,v1u,v1t,v2c,v2u,v2t;
        //coeffs in the inhomogeneous linear expression vic+viu*eu+vit*et, i=1,2,
        //of ongoing approximation to normalized input at the respective sample points
        //((2b+d)+i(2a+c))/((2b+d)-i(2a+c)), ((b+2d)+i(a+2c))/((b+2d)-i(a+2c))


struct complex temp;
struct edge *pnp;


pnp=pn;                 //initialize pointers
pnp++;

a=pc->a;                //initialize matrix entries
b=pc->b;
c=pc->c;
d=pc->d;
apc=a+c;
bpd=b+d;

pn->a=a;                //stuff matrix of first descendant
pn->b=b;
pn->c=apc;
pn->d=bpd;

pnp->a=apc;             //stuff matrix of second descendant
pnp->b=bpd;
pnp->c=c;
pnp->d=d;
```

```
alp=pc->alp;              //initialize lagged trig coeffs
bet=pc->bet;
gam=pc->gam;

e=pc->e;          //initialize wavelet coeff


temp.x=fn=fbar(-(b+bpd),a+apc);     //store new normalized input samples as the
temp.y=fnp=fbar(-(d+bpd),c+apc); //real and imag parts of temp to return

count+=2;                           //update count of total sample points

if(a>c){                            //prepare for stuffing alp,bet,gam in case a>c

        ax1=alp+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e;
        ay1=2*(b*b-a*a);

        bx1=bet+(c*d-a*d-b*c-a*b)*4*e;
        by1=4*a*b;

        cx1=gam+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e;
        cy1=-2*(a*a+b*b);

        ax2=alp+4*(d*d-c*c)*e;
        ay2=2*(c*c-d*d);

        bx2=bet+8*e*c*d;
        by2=-4*c*d;

        cx2=gam-4*e*(c*c+d*d);
        cy2=2*(c*c+d*d);

        chi=2*e+(                                   //chi=eu-et
        ((a+c)*(a+c)+(b+d)*(b+d))*(fc-gam)
        -alp*((b+d)*(b+d)-(a+c)*(a+c))
        -bet*2*(a+c)*(b+d)
        )/4.;
        }

else{                               //prepare for stuffing alp,bet,gam in case c>=a


       ax1=alp+4*e*(a*a-b*b);
       ay1=2*(b*b-a*a);

       bx1=bet-8*e*a*b;
       by1=4*a*b;

       cx1=gam+4*e*(a*a+b*b);
```

```
cy1=-2*(a*a+b*b);

ax2=alp+(a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e;
ay2=2*(c*c-d*d);

bx2=bet+(a*d-a*b+b*c+c*d)*4*e;
by2=-4*c*d;

cx2=gam+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e;
cy2=2*(c*c+d*d);



chi=-2*e+(                                  //chi=eu-et
((a+c)*(a+c)+(b+d)*(b+d))*(fc-gam)
-alp*((b+d)*(b+d)-(a+c)*(a+c))
-bet*2*(a+c)*(b+d)
)/4.;
}
```

//in any case, calculate the coeffs vic,viu,vit, for i=1,2

```
v1c=-fn+cx1+
(((b+bpd)*(b+bpd)-(a+apc)*(a+apc))*ax1
+2*(b+bpd)*(a+apc)*bx1)/((b+bpd)*(b+bpd)+(a+apc)*(a+apc));

v1t=cy1+
(((b+bpd)*(b+bpd)-(a+apc)*(a+apc))*ay1
+2*by1*(b+bpd)*(a+apc))/((b+bpd)*(b+bpd)+(a+apc)*(a+apc));

v1u=8./((b+bpd)*(b+bpd)+(a+apc)*(a+apc));

v2c=-fnp+cx2+
(((d+bpd)*(d+bpd)-(c+apc)*(c+apc))*ax2
+2*(d+bpd)*(c+apc)*bx2)/((d+bpd)*(d+bpd)+(c+apc)*(c+apc));

v2u=cy2+
(((d+bpd)*(d+bpd)-(c+apc)*(c+apc))*ay2
+2*(d+bpd)*(c+apc)*by2)/((d+bpd)*(d+bpd)+(c+apc)*(c+apc));

v2t=-8./((d+bpd)*(d+bpd)+(c+apc)*(c+apc));

sig1=v1c+chi*v1u;
tau1=v1t+v1u;

sig2=v2c+chi*v2u;
tau2=v2t+v2u;

et=-(sig1*tau1+sig2*tau2)/(tau1*tau1+tau2*tau2);     //calculate et and eu
eu=chi+et;
```

```
        pn->e=eu;                                   //stuff new wavelet coeffs
        pnp->e=et;

        pn->alp=ax1+ay1*et;          //stuff first lagged trig coeffs
        pn->bet=bx1+by1*et;
        pn->gam=cx1+cy1*et;

        pnp->alp=ax2+ay2*eu;         //stuff second lagged trig coeffs
        pnp->bet=bx2+by2*eu;
        pnp->gam=cx2+cy2*eu;

        return (temp);
}


struct complex bgener(struct edge *pc,struct edge *pn,double fc){

                //bgener(p,twofer,fc) stuffs twofer with the descendants
                //of p for the bottom of the circle, where fc is the current
                //normalized input sample and returns a complex number,
                //the real and imag parts being the normalized input values
                //required for the least-squares regularization

                //this is entirely analogous to tgener, and only the differences
                //will be noted here

        double fbar(int,int);

        int a,b,c,d,bpd,apc;
        double e,alp,bet,gam;

        double ax1,ay1,ax2,ay2;
        double bx1,by1,bx2,by2;
        double cx1,cy1,cx2,cy2;

        double sig1,tau1,sig2,tau2,chi;
        double v1c,v1u,v1t,v2c,v2u,v2t;

        double eu,et,fn,fnp;

        struct complex temp;

        struct edge *pnp;

        pnp=pn;
        pnp++;
```

```
a=pc->d;                    //differs from tgener in that d,-c,-b,a replaces a,b,c,d
b=-(pc->c);
c=-(pc->b);
d=pc->a;

apc=a+c;
bpd=b+d;

pn->a=bpd;                  //differs from tgener in that d,-c,-b,a replaces a,b,c,d
pn->b=-apc;
pn->c=-b;
pn->d=a;

pnp->a=d;                   //differs from tgener in that d,-c,-b,a replaces a,b,c,d
pnp->b=-c;
pnp->c=-bpd;
pnp->d=apc;

alp=pc->alp;
bet=pc->bet;
gam=pc->gam;


e=pc->e;


temp.x=fn=fbar(a+apc,b+bpd);      //differs from tgener in that
temp.y=fnp=fbar(c+apc,d+bpd);     //(-q,p) replaces (p,q)

count+=2;

if(a>c){

        ax1=alp+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e;
        ay1=2*(b*b-a*a);

        bx1=bet+(c*d-a*d-b*c-a*b)*4*e;
        by1=4*a*b;

        cx1=gam+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e;
        cy1=-2*(a*a+b*b);

        ax2=alp+4*(d*d-c*c)*e;
        ay2=2*(c*c-d*d);

        bx2=bet+8*e*c*d;
        by2=-4*c*d;

        cx2=gam-4*e*(c*c+d*d);
        cy2=2*(c*c+d*d);
```

```
chi=2*e+(
((a+c)*(a+c)+(b+d)*(b+d))*(fc-gam)
-alp*((b+d)*(b+d)-(a+c)*(a+c))
-bet*2*(a+c)*(b+d)
)/4.;

}


else{

ax1=alp+4*e*(a*a-b*b);
ay1=2*(b*b-a*a);

bx1=bet-8*e*a*b;
by1=4*a*b;

cx1=gam+4*e*(a*a+b*b);
cy1=-2*(a*a+b*b);

ax2=alp+(a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e;
ay2=2*(c*c-d*d);

bx2=bet+(a*d-a*b+b*c+c*d)*4*e;
by2=-4*c*d;

cx2=gam+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e;
cy2=2*(c*c+d*d);



chi=-2*e+(
((a+c)*(a+c)+(b+d)*(b+d))*(fc-gam)
-alp*((b+d)*(b+d)-(a+c)*(a+c))
-bet*2*(a+c)*(b+d)
)/4.;
}

v1c=-fn+cx1+
(((b+bpd)*(b+bpd)-(a+apc)*(a+apc))*ax1
+2*(b+bpd)*(a+apc)*bx1)/((b+bpd)*(b+bpd)+(a+apc)*(a+apc));

v1t=cy1+
(((b+bpd)*(b+bpd)-(a+apc)*(a+apc))*ay1
+2*by1*(b+bpd)*(a+apc))/((b+bpd)*(b+bpd)+(a+apc)*(a+apc));

v1u=8./((b+bpd)*(b+bpd)+(a+apc)*(a+apc));
```

```
        v2c=-fnp+cx2+
        (((d+bpd)*(d+bpd)-(c+apc)*(c+apc))*ax2
        +2*(d+bpd)*(c+apc)*bx2)/((d+bpd)*(d+bpd)+(c+apc)*(c+apc));

        v2u=cy2+
        (((d+bpd)*(d+bpd)-(c+apc)*(c+apc))*ay2
        +2*(d+bpd)*(c+apc)*by2)/((d+bpd)*(d+bpd)+(c+apc)*(c+apc));

        v2t=-8./((d+bpd)*(d+bpd)+(c+apc)*(c+apc));

        sig1=v1c+chi*v1u;
        tau1=v1t+v1u;

        sig2=v2c+chi*v2u;
        tau2=v2t+v2u;

        et=-(sig1*tau1+sig2*tau2)/(tau1*tau1+tau2*tau2);
        eu=chi+et;

        pn->e=eu;
        pnp->e=et;

        pn->alp=ax1+ay1*et;;
        pn->bet=bx1+by1*et;
        pn->gam=cx1+cy1*et;

        pnp->alp=ax2+ay2*eu;

        pnp->bet=bx2+by2*eu;
        pnp->gam=cx2+cy2*eu;

        return (temp);

}


void tprune(struct edge *p){                    //this routine implements the processing of terminal
                                                //edge p in the cascade for the top of the circle

        double alp, bet, gam,e;                 //local copies of lagged trig coeffs and wavelet coeff

        double alp1,alp2,bet1,bet2,gam1,gam2; //trig coeffs of the trigonometric function sigma at
                                                //the first and second (counter-clockwise) samples

        struct trip mob(struct complex,struct complex, struct complex,
                                        double,double,double);
                //mob(z1,z2,z3,f1,f2,f3) returns the triple with entries alp,bet,gam
                //where alp*cos theta + bet*sin theta + gam takes the respective values
                // f1,f2,f3 at the points z1,z2,z3 on the circle given as complex numbers
```

```
double fbar(int,int);                          //returns normalized input values

void fixup(struct trip,int,int,int,int,int);   //adjusts the output Fourier coeffs using sigma-tau

struct complex makecpx(int,int);               //input p,q returns (p-iq)/(p+iq)

void sixupp(struct trip,int,int,int,int,int);  //adjusts the 0,+1,-1 Fourier coeffs provided
                                               //SMODE=1

int a,b,c,d;                  //local variables for the matrix entries

struct trip trip1,trip2;      //first and second trig triples

a=p->a;                       //stuff local variables
b=p->b;
c=p->c;
d=p->d;
e=p->e;
alp=p->alp;
bet=p->bet;
gam=p->gam;

if (a>c){ //final update of trig fields in case a>c
        alp1=alp+e*2*(a*(a+c)+d*(d-b)-b*(b+d)-c*(c-a));
        bet1=bet+e*4*(c*(d-b)-a*(d+b));
        gam1=gam+e*2*(a*(a+c)+c*(a-c)+b*(b+d)-d*(d-b));

        alp2=alp+e*4*(d-c)*(d+c);
        bet2=bet+e*8*c*d;
        gam2=gam-e*4*(c*c+d*d);
        }

else{           //final update of trig fields in case c>=a
        alp1=alp+e*4*(a-b)*(a+b);
        bet1=bet-e*8*a*b;
        gam1=gam+e*4*(a*a+b*b);


        alp2=alp+e*2*(a*(a-c)+b*(d-b)-c*(c+a)+d*(b+d));
        bet2=bet+e*4*(a*(d-b)+c*(b+d));
        gam2=gam+e*2*(a*(a-c)-c*(a+c)-b*(d-b)-d*(b+d));
        }

trip1=mob(                    //calculate sigma for first descendant
        makecpx(-(3*b+d),3*a+c),
        makecpx(-(2*b+d),2*a+c),
        makecpx(-(3*b+2*d),3*a+2*c),
        fbar(-(3*b+d),3*a+c),
        fbar(-(2*b+d),2*a+c),
```

```
                fbar(-(3*b+2*d),3*a+2*c)
                );

        trip2=mob(                          //calculate sigma for second descendant
                makecpx(-(2*b+3*d),2*a+3*c),
                makecpx(-(b+2*d),a+2*c),
                makecpx(-(b+3*d),a+3*c),
                fbar(-(2*b+3*d),2*a+3*c),
                fbar(-(b+2*d),a+2*c),
                fbar(-(b+3*d),a+3*c)
                );

        count+=6;                           //update count of total sample points

        trip1.alp=(trip1.alp-alp1)/COR;     //difference of first trig functions
        trip1.bet=(trip1.bet-bet1)/COR;
        trip1.gam=(trip1.gam-gam1)/COR;

        trip2.alp=(trip2.alp-alp2)/COR;     //difference of second trig functions
        trip2.bet=(trip2.bet-bet2)/COR;
        trip2.gam=(trip2.gam-gam2)/COR;

        fixup(trip1,b+d,a+c,b,a,1); //adjust Fourier coeffs with first trig function difference

        fixup(trip2,d,c,b+d,a+c,1); //adjust Fourier coeffs with second trig function difference

        if(SMODE){                          //if SMODE=1, then adjust 0,+1,-1 Fourier coeffs as well
                sixupp(trip1,b+d,a+c,b,a,1);
                sixupp(trip2,d,c,b+d,a+c,1);
                }

}


void bprune(struct edge *p){                //this routine implements the processing of terminal
                                            //edge p in the cascade for the bottom of the circle

                                            //this is entirely analogous to tgener, and only the
                                            //differences will be noted here
        double alp, bet, gam,e;
        double alp1,alp2,bet1,bet2,gam1,gam2;
        struct trip mob(struct complex,struct complex, struct complex,
                                        double,double,double);
        double fbar(int,int);
        void fixup(struct trip, int,int,int,int,int);
        struct complex makecpx(int,int);
        void sixupp(struct trip,int,int,int,int,int);
        int a,b,c,d;
        struct trip trip1,trip2;
```

```
a=p->d;          //differs from tprune in that d,-c,-b,a replaces a,b,c,d
b=-(p->c);
c=-(p->b);
d=p->a;
e=p->e;
alp=p->alp;
bet=p->bet;
gam=p->gam;

if (a>c){
        alp1=alp+e*2*(a*(a+c)+d*(d-b)-b*(b+d)-c*(c-a));
        bet1=bet+e*4*(c*(d-b)-a*(d+b));
        gam1=gam+e*2*(a*(a+c)+c*(a-c)+b*(b+d)-d*(d-b));

        alp2=alp+e*4*(d-c)*(d+c);
        bet2=bet+e*8*c*d;
        gam2=gam-e*4*(c*c+d*d);
        }

else{
        alp1=alp+e*4*(a-b)*(a+b);
        bet1=bet-e*8*a*b;
        gam1=gam+e*4*(a*a+b*b);

        alp2=alp+e*2*(a*(a-c)+b*(d-b)-c*(c+a)+d*(b+d));
        bet2=bet+e*4*(a*(d-b)+c*(b+d));
        gam2=gam+e*2*(a*(a-c)-c*(a+c)-b*(d-b)-d*(b+d));
        }


trip1=mob(                      //arguments of mob differ from those in tprune
        makecpx(3*a+c,3*b+d),
        makecpx(2*a+c,2*b+d),
        makecpx(3*a+2*c,3*b+2*d),
        fbar(3*a+c,3*b+d),
        fbar(2*a+c,2*b+d),
        fbar(3*a+2*c,3*b+2*d)
        );

trip2=mob(                      //arguments of mob differ from those in tprune
        makecpx(2*a+3*c,2*b+3*d),
        makecpx(a+2*c,b+2*d),
        makecpx(a+3*c,b+3*d),
        fbar(2*a+3*c,2*b+3*d),
        fbar(a+2*c,b+2*d),
        fbar(a+3*c,b+3*d)
        );

count+=6;
```

```
        trip1.alp=(trip1.alp+alp1)/COR;     //differs from tprune in that alp,bet
        trip1.bet=(trip1.bet+bet1)/COR;     //is replaced by -alp,-bet
        trip1.gam=(trip1.gam-gam1)/COR;

        trip2.alp=(trip2.alp+alp2)/COR;     //differs from tprune in that alp,bet
        trip2.bet=(trip2.bet+bet2)/COR;     //is replaced by -alp,-bet
        trip2.gam=(trip2.gam-gam2)/COR;

        fixup(trip1,b+d,a+c,b,a,-1);        //last argument of fixup differs from tprune

        fixup(trip2,d,c,b+d,a+c,-1);Q       //last argument of fixup differs from tprune

        if(SMODE){                          //if SMODE=1, adjust the 0,+1,-1 Fourier coeffs

            sixupp(trip1,b+d,a+c,b,a,-1); //last argument of sixupp differs from tprune
            sixupp(trip2,d,c,b+d,a+c,-1); //last argument of sixupp differs from tprune

            }
}


struct trip mob(struct complex z1,struct complex z2,struct complex z3,
                        double f1,double f2,double f3){
        //mob(z1,z2,z3,f1,f2,f3) returns the triple with entries alp,bet,gam
        //where alp*cos theta + bet*sin theta + gam takes the respective values
        // f1,f2,f3 at the points z1,z2,z3 on the circle given as complex numbers

    double det,aa,bb,cc,dd;
    struct trip temp;

    aa=(z1.x-z3.x);
    bb=(z1.y-z3.y);
    cc=(z2.x-z3.x);
    dd=(z2.y-z3.y);

    det=aa*dd-bb*cc;//general principles guarantee that det is non-zero

    aa=aa/det;
    bb=bb/det;
    cc=cc/det;
    dd=dd/det;

    temp.alp=dd*(f1-f3)-bb*(f2-f3);
    temp.bet=aa*(f2-f3)-cc*(f1-f3);
    temp.gam=f3-z3.x*temp.alp-z3.y*temp.bet;

    return(temp);
}
```

```
void fixup(struct trip delt,int dm,int cm,int dp,int cp,int tb){

        //fixup delt,dm,cm,dp,cp,tb) adjusts the Fourier coeffs by adding the Fourier
        //coeffs of delt=T^C-T with support truncated to be the interval with endpoints
        //(dm+i*cm)/(dm-i*cm),  (dp+i*dp)/(dp-i*cp)

        //let tm, tp be the corresponding angles

        // tb=+1 on the top, and tb=-1 on the bottom of the circle

        int n;

        double cpp,cmp,cpm,cmm;
//in loop below on n, cpx=[cos(1+n)tx]/(n+1), cmx=[cos(1-n)tx]/(1-n),  for x=p,m

        double spm,smm, spp,smp;
//in loop below on n, spx=[sin(1+n)tx]/(1+n),  smx=[sin(1-n)tx]/(1-n),  for x=p,m

        double crm,crp,srm,srp;
//in loop below on n, crm=[cos n*tx]/n, srm= [sin n*tx]/n, for x=p,m

        double alp,bet,gam;             //local variables for entries of delt

        double ctp, ctm, cnp,cnm;  //in loop below on n, ctx=cos tx, cnx=cos n*tx, for x=p,m
        double stm,snm,stp,snp;         //in loop below on n, stx=sin  tx, snx=sin  n*tx, for x=p,m

        double cbufp,sbufp,cbufm,sbufm;
        struct complex makecpx(int,int),temp;


        alp=delt.alp;                   //stuff local variables
        bet=delt.bet;
        gam=delt.gam;

        temp=makecpx(dm,-cm);
        ctm=tb*temp.x;
        stm=tb*temp.y;

        temp=makecpx(dp,-cp);
        ctp=tb*temp.x;
        stp=tb*temp.y;

        cnp=ctp*ctp-stp*stp;
        snp=2*stp*ctp;

        cnm=ctm*ctm-stm*stm;
        snm=2*stm*ctm;
```

```
for(n=2;n<=N;n++){                    //loop over Fourier coeffs
        cbufp=ctp*cnp-stp*snp;        //update using formulas for cos
        sbufp=stp*cnp+ctp*snp;        //or sin of a sum of angles
        cbufm=ctm*cnm-stm*snm;
        sbufm=stm*cnm+ctm*snm;

        cpp=cbufp/(1+n);              //update for p
        cmp=(cbufp+2*stp*snp)/(1-n);
        spp=sbufp/(1+n);
        smp=(sbufp-2*ctp*snp)/(1-n);

        cpm=cbufm/(1+n);              //update for m
        cmm=(cbufm+2*stm*snm)/(1-n);
        spm=sbufm/(1+n);
        smm=(sbufm-2*ctm*snm)/(1-n);

        crp=2.*cnp/n;
        crm=2.*cnm/n;
        srp=2.*snp/n;
        srm=2.*snm/n;

        cnp=cbufp;                    //update for next iteration
        snp=sbufp;
        cnm=cbufm;
        snm=sbufm;

        fourier[N+n].x+=(             //adjust real part of nth Fourier coeff
        +alp*(smp+spp-smm-spm)
        +bet*(-cpp-cmp+cpm+cmm)
        +gam*(srp-srm)
        )/4.;

        fourier[N+n].y+=(             //adjust imag part of nth Fourier coeff
        +alp*(cpp-cmp-cpm+cmm)
        +bet*(spp-smp-spm+smm)
        +gam*(crp-crm)
        )/4.;

        }

    return;
}


void sixupp(struct trip triple,int dm,int cm,int dp,int cp,int tb){

//sixup(delt,dm,cm,dp,cp,tb) adjusts the 0,+1,-1 Fourier coeffs by adding the 0,+1,-1 Fourier
//coeffs of delt=T^C-T with support truncated to be the interval with endpoints
```

```
//(dm+i*cm)/(dm-i*cm).  (dp+i*dp)/(dp-i*cp)

//let tm, tp be the corresponding angles

// tb=+1 on the top, and tb=-1 on the bottom of the circle

        double alp,bet,gam;                 //local variables for entries of delt
        double com,sim,cop,sip,tm,tp;       //cox=cos tx, six=sin tx, for x=p,m

        struct complex temp;
        struct complex makecpx(int,int);


        temp=makecpx(dm,-cm);  //calculate com,sim,tm
        com=tb*temp.x;
        sim=tb*temp.y;
        tm=acos(com);

        temp=makecpx(dp,-cp);               //calculate cop,sip,tp
        cop=tb*temp.x;
        sip=tb*temp.y;
        tp=acos(cop);


        alp=tb*triple.alp;          //adjust alp,bet for the bottom of the circle
        bet=tb*triple.bet;
        gam=triple.gam;

        fourier[N].x+=(                     //update 0th Fourier coeff
                gam*(tp-tm)+alp*(sip-sim)-bet*(cop-com)
                )/2.;

        fourier[N+1].x+=(                   //update real part of 1st Fourier coeff
                        alp*(tp-tm)/2.
                        +gam*(sip-sim)
                        -bet*(cop*cop-sip*sip-com*com+sim*sim)/4.
                        +alp*(cop*sip-com*sim)/2.
                        )/2.;


                fourier[N+1].y+=(           //update imag part of 1st Fourier coeff
                        -bet*(tp-tm)/2.
                        +gam*(cop-com)
                        +alp*(cop*cop-sip*sip-com*com+sim*sim)/4.
                        +bet*(cop*sip-com*sim)/2.
                        )/2.;

}
```

```
//Here are the subroutines replacing tgener and bgener above in an
//implementation which takes advantage of renormalization

struct complex tgener(struct edge *pc,struct edge *pn,double fc)
{

    double fbar(int,int);
    int a,b,c,d,bpd,apc;
    double e,alp,bet,gam;
    double ax1,ay1,ax2,ay2;
    double bx1,by1,bx2,by2;
    double cx1,cy1,cx2,cy2;
    double sig1,tau1,sig2,tau2,chi;
    double v1c,v1u,v1t,v2c,v2u,v2t;
    double deriv(int,int,int,int,int,int);
    double dc,dn,dnp;
    double eu,et,fn,fnp;
    struct complex temp;
    struct edge *pnp;
    extern int count;

    pnp=pn;
    pnp++;


    a=pc->a;
    b=pc->b;
    c=pc->c;
    d=pc->d;

    apc=a+c;
    bpd=b+d;

    pn->a=a;
    pn->b=b;
    pn->c=apc;
    pn->d=bpd;

    pnp->a=apc;
    pnp->b=bpd;
    pnp->c=c;
    pnp->d=d;
```

```
alp=pc->alp;
bet=pc->bet;
gam=pc->gam;

e=pc->e;


temp.x=fn=fbar(-(b+bpd),a+apc);
temp.y=fnp=fbar(-(d+bpd),c+apc);
temp.n=0;

count+=2;

dc=deriv(a,b,c,d,-1,1);
dn=deriv(a,b,c,d,-1,2);
dnp=deriv(a,b,c,d,-2,1);

fc=fc/dc;
fn=fn/dn;
fnp=fnp/dnp;



if(a>c){
        ax1=alp+4.*e;
        bx1=bet-4.*e;
        cx1=gam;
        ax2=alp+4.*e;
        bx2=bet;
        cx2=gam-4.*e;
        chi=2.*((fc-gam-bet)/4.+e);
        }



else{


        ax1=alp+4.*e;
        bx1=bet;
        cx1=gam+4.*e;
        ax2=alp+4.*e;
        bx2=bet+4.*e;
        cx2=gam;
        chi=2.*((fc-gam-bet)/4.-e);
        }

v1c=-fn+cx1+(4.*bx1-3.*ax1)/5.;
v1t=-4./5.;
v1u=8./5.;
```

```
        v2c=-fnp+cx2+(4.*cx2+3.*ax2)/5.;
        v2u=4./5.;
        v2t=-8./5.;

        sig1=v1c+chi*v1u;
        tau1=v1t+v1u;

        sig2=v2c+chi*v2u;
        tau2=v2t+v2u;

        et=-(sig1*tau1*(1)+sig2*tau2*(1))/(tau1*tau1*(1)
        +tau2*tau2*(1));

        eu=chi+et;

        pn->e=eu;
        pnp->e=et;

        alp=ax1-2.*et;
        bet=bx1;
        gam=cx1-2.*et;

        pn->alp=(2.*bet+alp+gam)/2.;
        pn->bet=(bet-alp+gam);
        pn->gam=(2.*bet-alp+3.*gam)/2.;

        alp=ax2-2.*eu;
        bet=bx2;
        gam=cx2+2.*eu;

        pnp->alp=(-2.*bet+alp-gam)/2.;
        pnp->bet=alp+bet+gam;
        pnp->gam=(2.*bet+alp+3.*gam)/2.;


        return (temp);

}

double deriv(int a,int b,int c,int d,int p,int q){

        double x,y,nep;

        x=p*p-q*q;
        y=-2*p*q;
        x=x/(p*p+q*q);
        y=y/(p*p+q*q);
        nep=-(a*a+b*b+c*c+d*d)+x*(a*a+b*b-c*c-d*d)-y*(2*a*c+2*b*d);
        return(-2./nep);
```

```
}

struct complex bgener(struct edge *pc,struct edge *pn,double fc)
{
        double fbar(int,int);
        int a,b,c,d,bpd,apc;
        double e,alp,bet,gam;
        double ax1,ay1,ax2,ay2;
        double bx1,by1,bx2,by2;
        double cx1,cy1,cx2,cy2;
        double sig1,tau1,sig2,tau2,chi;
        double v1c,v1u,v1t,v2c,v2u,v2t;
        double eu,et,fn,fnp;
        extern int count;
        double deriv(int,int,int,int,int,int);
        double dc,dn,dnp;
        struct complex temp;
        struct edge *pnp;

        pnp=pn;
        pnp++;

        a=pc->d;
        b=-(pc->c);
        c=-(pc->b);
        d=pc->a;

        apc=a+c;
        bpd=b+d;

        pn->a=bpd;
        pn->b=-apc;
        pn->c=-b;
        pn->d=a;

        pnp->a=d;
        pnp->b=-c;
        pnp->c=-bpd;
        pnp->d=apc;

        alp=pc->alp;
        bet=pc->bet;
        gam=pc->gam;

        e=pc->e;

        temp.x=fn=fbar(a+apc,b+bpd);
        temp.y=fnp=fbar(c+apc,d+bpd);
        temp.n=0;
```

```
count+=2;

dc=deriv(d,-c,-b,a,1,1);
dn=deriv(d,-c,-b,a,2,1);
dnp=deriv(d,-c,-b,a,1,2);

fc=fc/dc;
fn=fn/dn;
fnp=fnp/dnp;

if(a>c){
        ax1=alp+4.*e;
        bx1=bet-4.*e;
        cx1=gam;

        ax2=alp+4.*e;
        bx2=bet;
        cx2=gam-4.*e;

        chi=2.*((fc-gam-bet)/4.+e);
        }


else{
        ax1=alp+4*e;
        bx1=bet;
        cx1=gam+4*e;

        ax2=alp+4*e;
        bx2=bet+4*e;
        cx2=gam;

        chi=2*((fc-gam-bet)/4-e);
        }

v1c=-fn+cx1+(4.*bx1-3.*ax1)/5.;
v1t=-4./5.;
v1u=8./5.;

v2c=-fnp+cx2+(4.*bx2+3.*ax2)/5.;
v2u=4./5.;
v2t=-8./5.;

sig1=v1c+chi*v1u;
tau1=v1t+v1u;

sig2=v2c+chi*v2u;
tau2=v2t+v2u;
```

```
et=-(sig1*tau1*(1)...g2*tau2*(1))/(tau1*tau1*(1)
+tau2*tau2*(1));

eu=chi+et;
pn->e=eu;
pnp->e=et;

alp=ax1-2.*et;
bet=bx1;
gam=cx1-2.*et;

pn->alp=(2.*bet+alp+gam)/2.;
pn->bet=(bet-alp+gam);
pn->gam=(2.*bet-alp+3.*gam)/2.;

alp=ax2-2.*eu;
bet=bx2;
gam=cx2+2.*eu;

pnp->alp=(-2.*bet+alp-gam)/2.;
pnp->bet=alp+bet+gam;
pnp->gam=(2.*bet+alp+3.*gam)/2.;

return (temp);

}
```

```
//This is an implen...itation in C of a preferred embodiment
//of the invention for the calculation of the inverse Fourier transform


#include <stdio.h>

#define SCAT 50                 //SCAT=[exp sinh^(-1)(1/SCALE)]^2

#define N 50                    //input bandwidth

#define MINMODE 0               //minimum frequency of input

#define PI 3.141592653589793

#define CHUNK 5000              //chunk of memory allocated as required


struct complex{                 //a complex number
                double x;
                double y;
                };

struct edge{                    //a complex arrow-structure
        int a,b,c,d;
        struct complex e,alp,bet,gam;
        };

struct pqxy{            //the basic data type of the output, where
        int p,q;        //p,q corresponds to the complex number (p-iq)/(p+iq)
        double x,y;     //at which the output function takes the complex value x+iy
        };

struct complex pfourier[2*N+1];

struct complex *fourier=pfourier+N;     //the input Fourier coeffs
                                        //indexed -N,...,-1,0,1,...,N

struct complex czer,cone,cmon;          //the modified 0,+1,-1 Fourier coeffs

struct pqxy graf[CHUNK];        //output spatial values in
                                //clockwise-order around the circle
                                //starting from the complex number -1

int outcount=0;                 //running tally of the total
                                //number of output values


void main(void){

        void genertop(struct edge *,int);   //recursive routine for top of circle
        void generbot(struct edge *,int);   //recursive routine for bottom of circle

        void normalout(void);               //output normalization routine
        void normalin(void);                //input normalization routine


        struct complex cogi(struct edge *); //returns complex wavelet coeff of argument
```

```
struct complex ei,emu,emt,eu,et:    //wavelet coeffs on I,U^{-1},T^{-1},U,T

struct complex emtt,emtu,emut,emuu; //wavelet coeffs on T^{-2},
                                    //T^{-1}U^{-1},U^{-1}T^{-1},U^{-2}

struct edge start[1];               //initial edge for each of
                                    //the various recursive calls
int i;

                        //begin input data

for(i=0;i<=2*N;i++){
        pfourier[i].x=0.;
        pfourier[i].y=0.;
        }

fourier[5].x=.5;
fourier[-5].x=.5;

                        //end input data


normalin();                         //this stuffs czer,cone,cmon with the
                                    //modified 0,+1,-1 Fourier coefficients

graf[outcount].p=0;                 //initial normalized output value
graf[outcount].q=1;
graf[outcount].x=0.;
graf[outcount].y=0.;
outcount++;
```

//calculate the wavelet coeffs for the various required edges of small generation

```
start->a=start->d=1;
start->c=0;
start->b=-1;
emt=cogi(start);

start->a=start->d=1;
start->c=0;
start->b=1;
et=cogi(start);

start->a=start->d=1;
start->c=-1;
start->b=0;
emu=cogi(start);

start->a=start->d=1;
start->c=1;
start->b=0;
eu=cogi(start);

start->a=start->d=1;
start->b=start->c=0;
ei=cogi(start);
```

```
start->a=start->d=1;
start->b=0;
start->c=-2;
emuu=cogi(start);

start->a=start->d=1;
start->b=-2;
start->c=0;
emtt=cogi(start);

start->a=1;
start->b=start->c=-1;
start->d=2;
emut=cogi(start);

start->a=2;
start->d=1;
start->b=start->c=-1;
emtu=cogi(start);
```

//serially perform the recursions for the initial edges in counter-clockwise order

```
start->a=start->d=1;                                    //initialize
start->c=1;
start->b=0;
start->e=eu;
start->alp.x=2.*ei.x-2.*et.x;
start->alp.y=2.*ei.y-2.*et.y;
start->bet.x=2.*(emt.x-emu.x)-2.*ei.x;
start->bet.y=2.*(emt.y-emu.y)-2.*ei.y;
start->gam.x=2.*ei.x-2.*et.x;
start->gam.y=2.*ei.y-2.*et.y;

genertop(start,1);                                      //recursion for U

start->a=start->d=1;                                    //initialize
start->c=0;
start->b=1;
start->e=et;
start->alp.x=2.*ei.x-2.*eu.x;
start->alp.y=2.*ei.y-2.*eu.y;
start->bet.x=2.*(emt.x-emu.x)+2.*ei.x;
start->bet.y=2.*(emt.y-emu.y)+2.*ei.y;
start->gam.x=-2.*ei.x+2.*eu.x;
start->gam.y=-2.*ei.y+2.*eu.y;

genertop(start,1);                                      //recursion for T

start->a=start->d=1;                                    //initialize
start->c=0;
start->b=-2;
start->e=emtt;
start->alp.x=-(-2.*ei.x+2.*emu.x-4.*emt.x+2.*emut.x);
start->alp.y=-(-2.*ei.y+2.*emu.y-4.*emt.y+2.*emut.y);
```

```
start->bet.x=-(2.*e....-2.*emu.x+2.*emt.x);
start->bet.y=-(2.*ei.y-2.*emu.y+2.*emt.y);
start->gam.x=2.*ei.x-2.*emu.x+4.*emt.x-2.*emut.x;
start->gam.y=2.*ei.y-2.*emu.y+4.*emt.y-2.*emut.y;

generbot(start,1);                                  //recursion for T^{-2}

start->a=1;                                         //initialize
start->d=2;
start->b=start->c=-1;
start->e=emut;
start->alp.x=-(-2.*ei.x+2.*emu.x+2.*emt.x);
start->alp.y=-(-2.*ei.y+2.*emu.y+2.*emt.y);
start->bet.x=-(2.*ei.x-2.*emu.x-6.*emt.x+4.*emtt.x);
start->bet.y=-(2.*ei.y-2.*emu.y-6.*emt.y+4.*emtt.y);
start->gam.x=2.*ei.x-2.*emu.x-6.*emt.x+4.*emtt.x;
start->gam.y=2.*ei.y-2.*emu.y-6.*emt.y+4.*emtt.y;

generbot(start,1);                                  //recursion for U^{-1}T^{-1}


start->a=2;                                         //initialize
start->d=1;
start->b=start->c=-1;
start->e=emtu;
start->alp.x=-(-2.*ei.x+2.*emu.x+2.*emt.x);
start->alp.y=-(-2.*ei.y+2.*emu.y+2.*emt.y);
start->bet.x=-(-2.*ei.x+6.*emu.x+2.*emt.x-4.*emuu.x);
start->bet.y=-(-2.*ei.y+6.*emu.y+2.*emt.y-4.*emuu.y);
start->gam.x=-2.*ei.x+6.*emu.x+2.*emt.x-4.*emuu.x;
start->gam.y=-2.*ei.y+6.*emu.y+2.*emt.y-4.*emuu.y;

generbot(start,1);                                  //recursion for T^{-1}U^{-1}

start->a=start->d=1;                                //initialize
start->b=0;
start->c=-2;
start->e=emuu;
start->alp.x=-(-2.*ei.x-4.*emu.x+2.*emt.x+2.*emtu.x);
start->alp.y=-(-2.*ei.y-4.*emu.y+2.*emt.y+2.*emtu.y);
start->bet.x=-(-2.*ei.x-2.*emu.x+2.*emt.x);
start->bet.y=-(-2.*ei.y-2.*emu.y+2.*emt.y);
start->gam.x=-2.*ei.x-4.*emu.x+2.*emt.x+2.*emtu.x;
start->gam.y=-2.*ei.y-4.*emu.y+2.*emt.y+2.*emtu.y;

generbot(start,1);                                  //recursion for U^{-2}

//recursions complete, and graf is stuffed with normalized output data in counter-clockwise order

normalout();                                        //un-normalize by altering
                                                    //each output value using the
                                                    //modified 0,+1,-1 Fourier coeffs


printf("cmon=%e +(i) %e\n czer=%e +(i) %e\n cone=%e +(i) %e\n\n",
        cmon.x,cmon.y,czer.x,czer.y,cone.x,cone.y);

for (i=0;i<outcount;i++)
```

```
                printf("(%d  %d)  %e+(i)%e\n",
                        graf[i].p,graf[i].q,graf[i].x,graf[i].y);
}


void genertop(struct edge *p,int g){ //recursion for top of the circle
        struct edge twofer[2],*ped; //recursion generates twofer[2] from p, where
                                    //the entries of twofer are the descendants of p
        struct complex temp;
        struct complex makecpx(int,int);        //input p,q returns (p-iq)/(p+iq)
        double ct,st;                           //cos,sin for various angles
        void genertop(struct edge *,int);       //recursive function

        void tgener(struct edge *,struct edge *);       //tgener(p,twofer) stuffs twofer[2]
                                                        //with the descendants of p

        int subtend(struct edge *),pp,qq;               //returns 1 if both argument edges
                                                        //subtend angles less than SCALE,
                                                        //otherwise returns 0

        tgener(p,twofer);                               //tgener(p,twofer) stuffs twofer
                                                        //with the two descendants of p

        if(subtend(twofer)){                            //if both descendants subtend
                                                        //angles less than SCALE, then
                                                        //write two output values and
                                                        //terminate the recursion
                ped=twofer;
                pp=-(ped->d);                           //stuff output values for the first
                qq=ped->c;                              //(counter-clockwise) sample point
                graf[outcount].p=pp;
                graf[outcount].q=qq;
                temp=makecpx(pp,qq);
                ct=temp.x;
                st=temp.y;
                graf[outcount].x=ped->alp.x*ct+ped->bet.x*st
                        +ped->gam.x+(ped->e.x)*4./(pp*pp+qq*qq);
                graf[outcount].y=ped->alp.y*ct+ped->bet.y*st
                        +ped->gam.y+(ped->e.y)*4./(pp*pp+qq*qq);
                outcount++;                             //update number of output values

                ped++;
                pp=-(ped->d);                           //stuff output values for the second
                qq=ped->c;                              //(counter-clockwise) sample point
                graf[outcount].p=pp;
                graf[outcount].q=qq;
                temp=makecpx(pp,qq);
                ct=temp.x;
                st=temp.y;
                graf[outcount].x=ped->alp.x*ct+ped->bet.x*st
                        +ped->gam.x;
                graf[outcount].y=ped->alp.y*ct+ped->bet.y*st
                        +ped->gam.y;
                outcount++;                             //update number of output values

                return;                                 //terminate the recursion
                }

        genertop(twofer,g+1);                           //in the contrary case that one of
```

```
        genertop(twofer+1,g...);                //the descendant edges subtends a
                                                //large angle, then continue the
        }                                       //recursion in counter-clockwise sense

void generbot(struct edge *p,int g){            //recursion for the bottom of
                                                        //the circle, this is entirely
                                                        //analogous to genertop, and
        struct edge twofer[2],*ped;             //only the differences will
                                                        //be noted here
        struct complex temp, makecpx(int,int);
        double ct,st;
        void generbot(struct edge *,int);
        void bgener(struct edge *,struct edge *);
        int subtend(struct edge *),pp,qq;

        bgener(p,twofer);

        if(subtend(twofer)){

                ped=twofer;
                pp=-ped->b;                             //differs from genertop in that
                qq=ped->a;                              //b,a replaces d,c
                graf[outcount].p=pp;
                graf[outcount].q=qq;
                temp=makecpx(pp,qq);
                ct=temp.x;
                st=temp.y;
                graf[outcount].x=-ped->alp.x*ct-ped->bet.x*st           //differs from
                        +ped->gam.x+(ped->e.x)*4./(pp*pp+qq*qq); // genertop in that
                graf[outcount].y=-ped->alp.y*ct-ped->bet.y*st           //alp and bet are
                        +ped->gam.y+(ped->e.y)*4./(pp*pp+qq*qq); // multiplied by -1
                outcount++;

                ped++;
                pp=-ped->b;                                     //differs from genertop in
                qq=ped->a;                                      //that b,a replaces d,c
                graf[outcount].p=pp;
                graf[outcount].q=qq;
                temp=makecpx(pp,qq);
                ct=temp.x;
                st=temp.y;
                graf[outcount].x=-ped->alp.x*ct-ped->bet.x*st       //differs from genertop
                        +ped->gam.x;                                //in that alp and bet are
                graf[outcount].y=-ped->alp.y*ct-ped->bet.y*st       //multiplied by -1
                        +ped->gam.y;
                outcount++;

                return;
                }

        generbot(twofer,g+1);
        generbot(twofer+1,g+1);

}

struct complex mult(struct complex u,struct complex v){         //multiplication
        struct complex temp;                                    //of complex numbers
```

```
        temp.x=u.x*v.x-u.y*v.y;
        temp.y=u.x*v.y+v.x*u.y;

        return(temp);
}


struct complex makecpx(int p,int q){                    //input p,q returns
        struct complex temp;                            //the complex number
        double den;                                     //(p-iq)/(p+iq)

        den=p*p+q*q;

        temp.x=(p*p-q*q)/den;
        temp.y=-2*p*q/den;

        return(temp);
}

int subtend(struct edge *p){                            //subtend returns 1 if each of
        int nn;                                         //of the two edges in the
                                                        //array p subtends an angle
        nn=(p->a)*(p->c)+(p->b)*(p->d);                 //approximately less than
        if(nn*nn<SCAT)                                  //SCALE and returns 0 otherwise
                return(0);

        p++;

        nn=(p->a)*(p->c)+(p->b)*(p->d);

        if(nn*nn<SCAT)
                return(0);

        return(1);
}


void normalin(void){
                                                        //this routine calculates the modified
        struct complex cz[4],co[4],cm[4];               //0,+1,-1 Fourier coeffs czero,cone,cmon
                                                        //from the input Fourier coeffs

        struct complex temp;

        int n;

        cz[0].x=-1.;                                    //intialize array for czero calculation
        cz[0].y=0.;
        cz[1].x=0.;
        cz[1].y=0.;
        cz[2].x=-1.;
        cz[2].y=0.;
        cz[3].x=0.;
        cz[3].y=0.;

        co[0].x=0.;                                     //initialize array for cone calculation
        co[0].y=0.;
        co[1].x=-1.;
        co[1].y=0.;
```

```
co[2].x=0.;
co[2].y=1.;
co[3].x=0.;
co[3].y=0.;

cm[0].x=0.;                          //initialize array for cmon calculation
cm[0].y=0.;
cm[1].x=0.;
cm[1].y=0.;
cm[2].x=0.;
cm[2].y=-1.;
cm[3].x=-1.;
cm[3].y=0.;

czer=fourier[0];
cone=fourier[1];
cmon=fourier[-1];

for(n=2;n<=N;n++){

        temp=mult(fourier[n],cz[n%4]);
        czer.x+=-temp.x;              //contribution to czer from the
        czer.y+=-temp.y;              //positive Fourier coeffs

        temp=mult(fourier[-n],cz[(4*N-n)%4]);


        czer.x+=-temp.x;              //contribution to czer from the
        czer.y+=-temp.y;              //negative Fourier coeffs

        temp=mult(fourier[n],co[n%4]);

        cone.x+=-temp.x;          //contribution to cone from the
        cone.y+=-temp.y;          //positive Fourier coeffs

        temp=mult(fourier[-n],co[(4*N-n)%4]);

        cone.x+=-temp.x;          //contribution to cone from the
        cone.y+=-temp.y;          //negative Fourier coeffs

        temp=mult(fourier[n],cm[n%4]);

        cmon.x+=-temp.x;              //contribution to cmon from the
        cmon.y+=-temp.y;              //positive Fourier coeffs

        temp=mult(fourier[-n],cm[(4*N-n)%4]);

        cmon.x+=-temp.x;              //contribution to cmon from the
        cmon.y+=-temp.y;              //negative Fourier coeffs
        }
}


void normalout(void){                        //this routine alters the
    int n,m;                                 //output values in the
    struct complex temp,temp1,makecpx(int,int);  //array graf by adding
    double ct,st;                            //czero+cone*z+cmon*z^(-1),
```

```
                                                        //where z=(p-iq)/(p
        for (n=0;n<outcount;n++){

                temp=makecpx(graf[n].p,graf[n].q);
                temp1=mult(temp,cone);

                temp.y=-temp.y;
                temp=mult(temp,cmon);

                graf[n].x+=czer.x+temp1.x+temp.x;
                graf[n].y+=czer.y+temp1.y+temp.y;

                }
}

struct complex cogi(struct edge *p){        //cogi(p) calculates the wavelet coeff for the
                                            //arrow underlying edge p from the Fourier coeffs
        int a,b,c,d,n;
        struct complex makecpx(int,int), mult(struct complex,struct complex);
        struct complex temp2,temp1,temp,crun,cb1,cm1,cb2,cm2;

        a=p->a;
        b=p->b;
        c=p->c;
        d=p->d;
        crun.x=0.;
        crun.y=0.;

        cm1=makecpx(b,-a);
        cm2=makecpx(d,-c);

        cb1=cm1;
        cb2=cm2;

        for(n=2;n<=N;n++){
                cb1=mult(cb1,cm1);      //updated nth power of cm1
                cb2=mult(cb2,cm2);      //updated nth power of cm2

                if(n<MINMODE)//skip contribution to wavelet coeffs
                        continue;       //from low-frequency Fourier coeffs

                temp.x=n;
                temp.y=b*d+a*c;
                temp1=mult(cb1,temp);
                temp.y=-temp.y;
                temp2=mult(cb2,temp);
                temp.x=temp1.x+temp2.x;
                temp.y=temp1.y+temp2.y;
                temp1=mult(fourier[n],temp);
                crun.x+=temp1.x;                //contribution to wavelet coeff from
                crun.y+=temp1.y;                //nth Fourier coeff for n positive

                temp.x=-temp.x;
                temp2=mult(fourier[-n],temp);
                crun.x+=temp2.x;                //contribution to wavelet coeff from
                crun.y+=temp2.y;                //nth Fourier coeff for n negative

                }
```

```
        temp=crun;
        crun.x=-temp.y/4.;                    //overall multiple of i/4
        crun.y=temp.x/4.;

        return(crun);

}



void tgener(struct edge *pc,struct edge *pn){        //tgener(p,twofer) stuffs twofer
                                                     //with descendants of p for top
                                                     //of circle

        int a,b,c,d,bpd,apc;                  //matrix entries a,b,c,d,
                                              //bpd=b+d and apc=a+c

        struct complex e,alp,bet,gam;         //wavelet coefficient e, trig coefficients alp,bet,gam

        struct complex ax1,ax2;               //index 1,2 refers to first,second
        struct complex bx1,bx2;               //(counter-clockwise) descendants
        struct complex cx1,cx2;               //prefix a,b,c refers to alp,bet,gam
        int ay1,ay2,by1,by2,cy1,cy2;          //update procedure below defines x,y

        struct complex eu,et;                 //wavelet coeffs of first, second
                                              //descendant edges
        struct complex temp;

        struct complex cogi(struct edge *),mult(struct complex,struct complex);
        struct complex makecpx(int,int);

        double ct,st;

        struct edge *pnp;

        pnp=pn;                               //initialize pointers
        pnp++;

        a=pc->a;                              //initialize matrix entries
        b=pc->b;
        c=pc->c;
        d=pc->d;

        apc=a+c;
        bpd=b+d;

        pn->a=a;                              //stuff matrix of first descendant
        pn->b=b;
        pn->c=apc;
        pn->d=bpd;

        pnp->a=apc;                           //stuff matrix of descendant
        pnp->b=bpd;
        pnp->c=c;
        pnp->d=d;
```

```
alp=pc->alp;                    //initialize lagged trig coeffs
bet=pc->bet;
gam=pc->gam;

e=pc->e;                //initialize wavelet coeff

if(a>c){                        //prepare for stuffing alp,bet,gam in case a>c

        ax1.x=alp.x+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e.x;
        ax1.y=alp.y+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e.y;


        ay1=2*(b*b-a*a);

        bx1.x=bet.x+(c*d-a*d-b*c-a*b)*4*e.x;
        bx1.y=bet.y+(c*d-a*d-b*c-a*b)*4*e.y;

        by1=4*a*b;

        cx1.x=gam.x+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e.x;
        cx1.y=gam.y+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e.y;

        cy1=-2*(a*a+b*b);

        ax2.x=alp.x+4*(d*d-c*c)*e.x;
        ax2.y=alp.y+4*(d*d-c*c)*e.y;

        ay2=2*(c*c-d*d);

        bx2.x=bet.x+8*e.x*c*d;
        bx2.y=bet.y+8*e.y*c*d;

        by2=-4*c*d;

        cx2.x=gam.x-4*e.x*(c*c+d*d);
        cx2.y=gam.y-4*e.y*(c*c+d*d);

        cy2=2*(c*c+d*d);
        }

else{                           //prepare for stuffing alp,bet,gam in case c>=a

        ax1.x=alp.x+4*e.x*(a*a-b*b);
        ax1.y=alp.y+4*e.y*(a*a-b*b);

        ay1=2*(b*b-a*a);

        bx1.x=bet.x-8*e.x*a*b;
        bx1.y=bet.y-8*e.y*a*b;

        by1=4*a*b;

        cx1.x=gam.x+4*e.x*(a*a+b*b);
        cx1.y=gam.y+4*e.y*(a*a+b*b);

        cy1=-2*(a*a+b*b);

        ax2.x=alp.x+(a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e.x;
```

```
ax2.y=al[illegible](a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e.y;

ay2=2*(c*c-d*d);

bx2.x=bet.x+(a*d-a*b+b*c+c*d)*4*e.x;
bx2.y=bet.y+(a*d-a*b+b*c+c*d)*4*e.y;

by2=-4*c*d;


cx2.x=gam.x+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e.x;
cx2.y=gam.y+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e.y;

cy2=2*(c*c+d*d);
}

pn->e=eu=cogi(pn);                  //calculate wavelet coeffs
pnp->e=et=cogi(pnp);                //of the descendants

pn->alp.x=ax1.x+ay1*et.x;           //update first alp
pn->alp.y=ax1.y+ay1*et.y;

pn->bet.x=bx1.x+by1*et.x;           //update first bet
pn->bet.y=bx1.y+by1*et.y;

pn->gam.x=cx1.x+cy1*et.x;           //update first gam
pn->gam.y=cx1.y+cy1*et.y;

pnp->alp.x=ax2.x+ay2*eu.x;          //update second alp
pnp->alp.y=ax2.y+ay2*eu.y;

pnp->bet.x=bx2.x+by2*eu.x;          //update second bet
pnp->bet.y=bx2.y+by2*eu.y;

pnp->gam.x=cx2.x+cy2*eu.x;          //update second gam
pnp->gam.y=cx2.y+cy2*eu.y;


}


void bgener(struct edge *pc,struct edge *pn){   //bgener(p,twofer) stuffs twofer
                                                //with descendants of p for
                                                //bottom of circle,
                                                //it is entirely analogous to
                                                //tgener, and only the
                                                //differences will be noted here

    int a,b,c,d,bpd,apc;
    struct complex e,alp,bet,gam;


    struct complex ax1,ax2;
    struct complex bx1,bx2;
    struct complex cx1,cx2;

    int ay1,ay2,by1,by2,cy1,cy2;

    struct complex eu,et;
```

```
struct complex ten.,r ,

struct complex cogi(struct edge *),mult(struct complex,struct complex);
struct complex makecpx(int,int);

struct edge *pnp;

double ct,st;

pnp=pn;
pnp++;

a=pc->d;                //differs from tgener in that d,-c,-b,a
b=-(pc->c);             //replaces a,b,c,d
c=-(pc->b);
d=pc->a;

apc=a+c;
bpd=b+d;

pn->a=bpd;              //differs from tgener in that d,-c,-b,a
pn->b=-apc;             //replaces a,b,c,d
pn->c=-b;
pn->d=a;

pnp->a=d;               //differs from tgener in that d,-c,-b,a
pnp->b=-c;              //replaces a,b,c,d
pnp->c=-bpd;
pnp->d=apc;

alp=pc->alp;
bet=pc->bet;
gam=pc->gam;

e=pc->e;

if(a>c){

        ax1.x=alp.x+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e.x;
        ax1.y=alp.y+(d*d-c*c+2*(a*c-b*d)+a*a-b*b)*2*e.y;

        ay1=2*(b*b-a*a);

        bx1.x=bet.x+(c*d-a*d-b*c-a*b)*4*e.x;
        bx1.y=bet.y+(c*d-a*d-b*c-a*b)*4*e.y;

        by1=4*a*b;

        cx1.x=gam.x+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e.x;
        cx1.y=gam.y+(2*(a*c+b*d)-c*c-d*d+a*a+b*b)*2*e.y;

        cy1=-2*(a*a+b*b);

        ax2.x=alp.x+4*(d*d-c*c)*e.x;
        ax2.y=alp.y+4*(d*d-c*c)*e.y;

        ay2=2*(c*c-d*d);

        bx2.x=bet.x+8*e.x*c*d;
```

```
            bx2.y=bet.y-8*e.y*c*d;

            by2=-4*c*d;

            cx2.x=gam.x-4*e.x*(c*c+d*d);
            cx2.y=gam.y-4*e.y*(c*c+d*d);

            cy2=2*(c*c+d*d);
            }

else{
            ax1.x=alp.x+4*e.x*(a*a-b*b);
            ax1.y=alp.y+4*e.y*(a*a-b*b);

            ay1=2*(b*b-a*a);

            bx1.x=bet.x-8*e.x*a*b;
            bx1.y=bet.y-8*e.y*a*b;

            by1=4*a*b;

            cx1.x=gam.x+4*e.x*(a*a+b*b);
            cx1.y=gam.y+4*e.y*(a*a+b*b);

            cy1=-2*(a*a+b*b);

            ax2.x=alp.x+(a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e.x;
            ax2.y=alp.y+(a*a-b*b+2*(b*d-a*c)-c*c+d*d)*2*e.y;

            ay2=2*(c*c-d*d);

            bx2.x=bet.x+(a*d-a*b+b*c+c*d)*4*e.x;
            bx2.y=bet.y+(a*d-a*b+b*c+c*d)*4*e.y;

            by2=-4*c*d;

            cx2.x=gam.x+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e.x;
            cx2.y=gam.y+(a*a+b*b-2*(a*c+b*d)-c*c-d*d)*2*e.y;

            cy2=2*(c*c+d*d);
            }


pn->e=eu=cogi(pn);
pnp->e=et=cogi(pnp);

pn->alp.x=ax1.x+ay1*et.x;
pn->alp.y=ax1.y+ay1*et.y;

pn->bet.x=bx1.x+by1*et.x;
pn->bet.y=bx1.y+by1*et.y;

pn->gam.x=cx1.x+cy1*et.x;
pn->gam.y=cx1.y+cy1*et.y;

pnp->alp.x=ax2.x+ay2*eu.x;
pnp->alp.y=ax2.y+ay2*eu.y;

pnp->bet.x=bx2.x+by2*eu.x;
```

```
        pnp->bet.y=bx2.y+[illegible]*eu.y;

        pnp->gam.x=cx2.x+cy2*eu.x;
        pnp->gam.y=cx2.y+cy2*eu.y;
}



        //Here are the subroutines replacing tgener and bgener above in an
        //implementation which takes advantage of renormalization

void tgener(struct edge *pc,struct edge *pn)
{

        int a,b,c,d,bpd,apc;
        struct complex e,alp,bet,gam;

        extern int outcount;
        extern struct pqxy graf[CHUNK];

        struct complex ax1,ax2;
        struct complex bx1,bx2;
        struct complex cx1,cx2;
        struct complex eu,et;
        struct complex cogi(struct edge *);

        double ct,st;
        struct edge *pnp;

        pnp=pn;
        pnp++;
        a=pc->a;
        b=pc->b;
        c=pc->c;
        d=pc->d;

        apc=a+c;
        bpd=b+d;

        pn->a=a;
        pn->b=b;
        pn->c=apc;
        pn->d=bpd;

        pnp->a=apc;
        pnp->b=bpd;
        pnp->c=c;
        pnp->d=d;

        alp=pc->alp;
        bet=pc->bet;
        gam=pc->gam;

        e=pc->e;
```

```
if(a>c){
        ax1.x=alp.x+4.*e.x;
        ax1.y=alp.y+4.*e.y;
        bx1.x=bet.x-4.*e.x;
        bx1.y=bet.y-4.*e.y;
        cx1.x=gam.x;
        cx1.y=gam.y;

        ax2.x=ax1.x;
        ax2.y=ax1.y;
        bx2.x=bet.x;
        bx2.y=bet.y;
        cx2.x=gam.x-4.*e.x;
        cx2.y=gam.y-4.*e.y;
        }


else{
        ax1.x=alp.x+4.*e.x;
        ax1.y=alp.y+4.*e.y;
        bx1.x=bet.x;
        bx1.y=bet.y;
        cx1.x=gam.x+4.*e.x;
        cx1.y=gam.y+4.*e.y;

        ax2.x=ax1.x;
        ax2.y=ax1.y;
        bx2.x=bet.x+4.*e.x;
        bx2.y=bet.y+4.*e.y;
        cx2.x=gam.x;
        cx2.y=gam.y;
        }


pn->e=eu=cogi(pn);
pnp->e=et=cogi(pnp);

alp.x=ax1.x-2.*et.x;
bet.x=bx1.x;
gam.x=cx1.x-2.*et.x;

alp.y=ax1.y-2.*et.y;
bet.y=bx1.y;
gam.y=cx1.y-2.*et.y;

pn->alp.x=(2.*bet.x+alp.x+gam.x)/2.;
pn->bet.x=(bet.x-alp.x+gam.x);
pn->gam.x=(2.*bet.x-alp.x+3.*gam.x)/2.;

pn->alp.y=(2.*bet.y+alp.y+gam.y)/2.;
pn->bet.y=(bet.y-alp.y+gam.y);
pn->gam.y=(2.*bet.y-alp.y+3.*gam.y)/2.;

alp.x=ax2.x-2.*eu.x;
bet.x=bx2.x;
gam.x=cx2.x+2.*eu.x;

alp.y=ax2.y-2.*eu.y;
```

```
        bet.y=bx2.y;
        gam.y=cx2.y+2.*eu.y;

        pnp->alp.x=(-2.*bet.x+alp.x-gam.x)/2.;
        pnp->bet.x=alp.x+bet.x+gam.x;
        pnp->gam.x=(2.*bet.x+alp.x+3.*gam.x)/2.;

        pnp->alp.y=(-2.*bet.y+alp.y-gam.y)/2.;
        pnp->bet.y=alp.y+bet.y+gam.y;
        pnp->gam.y=(2.*bet.y+alp.y+3.*gam.y)/2.;

}

double deriv(int a,int b,int c,int d,int p,int q){

        double x,y,nep;

        x=p*p-q*q;
        y=-2*p*q;
        x=x/(p*p+q*q);
        y=y/(p*p+q*q);
        nep=-(a*a+b*b+c*c+d*d)+x*(a*a+b*b-c*c-d*d)-y*2.*(a*c+b*d);
        return(-2./nep);

}

void bgener(struct edge *pc,struct edge *pn)
{


        int a,b,c,d,bpd,apc;
        struct complex e,alp,bet,gam;
        extern int outcount;
        extern struct pqxy graf[CHUNK];
        struct complex ax1,ax2;
        struct complex bx1,bx2;
        struct complex cx1,cx2;
        struct complex eu,et;
        struct complex cogi(struct edge *);
        double ct,st;
        struct edge *pnp;

        pnp=pn;
        pnp++;


        a=pc->d;
        b=-(pc->c);
        c=-(pc->b);
        d=pc->a;

        apc=a+c;
        bpd=b+d;

        pn->a=bpd;
        pn->b=-apc;
        pn->c=-b;
        pn->d=a;
```

```
pnp->a=d;
pnp->b=-c;
pnp->c=-bpd;
pnp->d=apc;

alp=pc->alp;
bet=pc->bet;
gam=pc->gam;

e=pc->e;


if(a>c){
        ax1.x=alp.x+4.*e.x;
        ax1.y=alp.y+4.*e.y;
        bx1.x=bet.x-4.*e.x;
        bx1.y=bet.y-4.*e.y;
        cx1.x=gam.x;
        cx1.y=gam.y;

        ax2.x=ax1.x;
        ax2.y=ax1.y;
        bx2.x=bet.x;
        bx2.y=bet.y;
        cx2.x=gam.x-4.*e.x;
        cx2.y=gam.y-4.*e.y;
        }


else{
        ax1.x=alp.x+4.*e.x;
        ax1.y=alp.y+4.*e.y;
        bx1.x=bet.x;
        bx1.y=bet.y;
        cx1.x=gam.x+4.*e.x;
        cx1.y=gam.y+4.*e.y;

        ax2.x=ax1.x;
        ax2.y=ax1.y;
        bx2.x=bet.x+4.*e.x;
        bx2.y=bet.y+4.*e.y;
        cx2.x=gam.x;
        cx2.y=gam.y;
        }

pn->e=eu=cogi(pn);
pnp->e=et=cogi(pnp);

pn->e=eu;
pnp->e=et;

alp.x=ax1.x-2.*et.x;
bet.x=bx1.x;
gam.x=cx1.x-2.*et.x;

alp.y=ax1.y-2.*et.y;
bet.y=bx1.y;
```

```
gam.y=cx1.y-2.*e..,

pn->alp.x=(2.*bet.x+alp.x+gam.x)/2.;
pn->bet.x=(bet.x-alp.x+gam.x);
pn->gam.x=(2.*bet.x-alp.x+3.*gam.x)/2.;

pn->alp.y=(2.*bet.y+alp.y+gam.y)/2.;
pn->bet.y=(bet.y-alp.y+gam.y);
pn->gam.y=(2.*bet.y-alp.y+3.*gam.y)/2.;


alp.x=ax2.x-2.*eu.x;
bet.x=bx2.x;
gam.x=cx2.x+2.*eu.x;

alp.y=ax2.y-2.*eu.y;
bet.y=bx2.y;
gam.y=cx2.y+2.*eu.y;

pnp->alp.x=(-2.*bet.x+alp.x-gam.x)/2.;
pnp->bet.x=alp.x+bet.x+gam.x;
pnp->gam.x=(2.*bet.x+alp.x+3.*gam.x)/2.;

pnp->alp.y=(-2.*bet.y+alp.y-gam.y)/2.;
pnp->bet.y=alp.y+bet.y+gam.y;
pnp->gam.y=(2.*bet.y+alp.y+3.*gam.y)/2.;
}
```

Methods of D al Filtering and Multi-Dimensional Data ompression
Using the Farey Quadrature and Arithmetic, Fan, and Modular Wavelets

Claims

I claim:

1. A method of compressing and reconstructing input data comprising the steps of: providing input data in an initial form; sampling the input data at the points of the Farey quadrature; transforming the Farey-sampled data into an intermediate form; quantizing the Farey-sampled data while in the intermediate form; using the quantized intermediate form to store or transmit the Farey-sampled data; de-quantizing the quantized intermediate form of the Farey-sampled data prior to reconstructing the data; and reconstructing the intermediate form of the Farey-sampled data in order to obtain reconstructed data for use.

2. A method as recited in claim 1 wherein: the step of transforming the Farey-sampled data into an intermediate form is achieved via a wavelet transformation; and the intermediate form of the Farey-sampled data takes the form of wavelet coefficients.

3. A method as recited in claim 2 wherein the wavelet transformation is based on arithmetic wavelets.

4. A method as recited in claim 3 wherein the arithmetic wavelets are defined in accordance with the following expressions:

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \le \theta \le \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \le \theta \le \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \le \theta \le \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \le \theta \le 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetic arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a top ; hmetic arrow and $c \geq a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \leq \theta \leq \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \leq \theta \leq \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \leq \theta \leq \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \leq \theta \leq \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \leq \theta \leq \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \leq \theta \leq \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \geq a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \leq \theta \leq \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \leq \theta \leq \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \leq \theta \leq \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

5. A method as reci in claim 2 wherein the wavelet transfc ation is based on modular wavelets.

6. A method as recited in claim 5 wherein the modular wavelets are defined in accordance with the following expressions:

$$\psi_A(\theta) = \sum_{n\geq 0} n\ \overline{\vartheta}_{U^n A}(\theta),$$
$$\psi_{SA}(\theta) = \sum_{n\geq 0} n\ \overline{\vartheta}_{T^{-n} A}(\theta),$$

where

$$\overline{\vartheta}_A(\theta) = \begin{cases} \tilde{\vartheta}_A(\theta); & \text{if } A \neq U^{-1}, T^{-1}, \\ \tilde{\vartheta}_{U^{-1}}(\theta) - 2\sin\theta; & \text{if } A = U^{-1}, \\ \tilde{\vartheta}_{T^{-1}}(\theta) + 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$; and

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \leq \theta \leq \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \leq \theta \leq \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \leq \theta \leq \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \leq \theta \leq 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetric arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \leq \theta \leq \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \leq \theta \leq \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \leq \theta \leq \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \geq a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

7. A method as recite · ı claim 2 wherein the wavelet transform. ɔn is based on fan wavelets.

8. A method as recited in claim 7 wherein the fan wavelets are defined in accordance with the following expressions:

$$\phi_A(\theta) = \sum_{n\geq 0} \overline{\vartheta}_{U^n A}(\theta),$$
$$\phi_{SA}(\theta) = \sum_{n\geq 0} \overline{\vartheta}_{T^{-n} A}(\theta),$$

where

$$\overline{\vartheta}_A(\theta) = \begin{cases} \tilde{\vartheta}_A(\theta); & \text{if } A \neq U^{-1}, T^{-1}, \\ \tilde{\vartheta}_{U^{-1}}(\theta) - 2\sin\theta; & \text{if } A = U^{-1}, \\ \tilde{\vartheta}_{T^{-1}}(\theta) + 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$; and

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \leq \theta \leq \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \leq \theta \leq \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \leq \theta \leq \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \leq \theta \leq 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetric arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \leq \theta \leq \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \leq \theta \leq \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \leq \theta \leq \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \geq a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

9. A method as recited claim 2 wherein the steps of sampling tl nput data at the points of the Farey quadrature and transforming the Farey-sampled data into wavelet coefficients is characterized by a binary cascade of arrow structures arising from the calculation of wavelet coefficients by sampling the input data in its initial form at the points of the Farey quadrature as defined along a circle in the complex plane.

10. A method as recited in claim 2 wherein the wavelet transformation is based on arithmetic wavelets and the step of transforming the Farey-sampled data into arithmetic wavelet coefficients includes the step of regularization.

11. A method as recited in claim 2 wherein the wavelet transformation and the reconstruction are based on a finite number of wavelets.

12. A method as recited in claim 2 wherein: wavelet transformation is achieved via a plurality of wavelet transformation operations each for a specified arc along a circle in the complex plane; and the plurality of wavelet transformation operations are calculated in parallel with each other.

13. A method as recited in claim 1 wherein the quantized intermediate form of the data is stored and retrieved prior to de-quantizing and reconstructing the data for use.

14. A method as recited in claim 1 wherein: the steps of transforming the Farey-sampled data into the intermediate form and quantizing the intermediate form of the Farey-sampled data occur at a first physical location; the quantized intermediate form of the data is transmitted to a second physical location; and the steps of de-quantizing the quantized intermediate form of the Farey-sampled data and reconstructing the intermediate form of the Farey-sampled data in order to obtain reconstructed data occurs at the second physical location.

15. A method as recited in claim 14 further comprising the step of transmitting a reconstruction algorithm to the second physical location in order to accomplish reconstruction at the second physical location.

16. A method as recited in claim 1 wherein the input data is digital data.

17. A method as recited in claim 1 wherein the input data is analog data and the analog input data is sampled by placing analog sensors at locations corresponding to the points of the Farey quadrature.

18. A method as recited in claim 1 wherein the input data and the reconstructed data are multi-dimensional.

19. A method as recited in claim 2 wherein the step of reconstructing the intermediate form of the Farey-sampled data in order to obtain the reconstructed data is achieved via an inverse wavelet transformation which is characterized by a binary cascade.

20. A method as recited in claim 2 wherein the calculation of the wavelet transformation

ıncludes the step of rei :nalization.

21. A method as recited in claim 1 wherein the sampling of data at the points of the Farey quadrature includes temporary storage of sequences of data in a buffer.

22. A digital signal processing method for obtaining data in a transform domain comprising the steps of: providing input data in an initial form; sampling the input data at the points of the Farey quadrature; using an intermediate transform to transform the Farey-sampled data into an intermediate form; and using a primary transform to convert the data in intermediate form of the Farey-sampled data into transform coefficients representative of the data in a transform domain.

23. A method as recited in claim 22 wherein the primary transformation consists of one of the groups of following transformations: Fourier, Hilbert, Haar, Laplace, Bessel, Laguerre, Hermite, Chebyshev, Hotelling, Mersenne and Fermat.

24. A method as recited in claim 22 wherein the primary transformation consists of the Fourier transform, and the intermediate transform is a wavelet transformation based on arithmetic wavelets.

25. A method as recited in claim 24 wherein the intermediate transform is an arithmetic wavelet transform and the primary transform is a Fourier transform, and the conversion of the data from the intermediate form to the Fourier coefficients, c_n^A, is given by the following expressions:

$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetic arrow; and

$$\begin{aligned} \pi\, i\, (n^3 - n)\ \ c_n^A \ \ = \ \ & -[(c-a)^2 + (b-d)^2]\left[\frac{(b-d) - i(a-c)}{(b-d) + i(a-c)}\right]^n \\ & + 2(c^2+d^2)\left[\frac{d-ic}{d+ic}\right]^n \ + \ 2(a^2+b^2)\left[\frac{b-ia}{b+ia}\right]^n \\ & - [(c+a)^2 + (b+d)^2]\left[\frac{(b+d) - i(a+c)}{(b+d) + i(a+c)}\right]^n; \end{aligned}$$

and the coefficients c_0^A, c_1^A, c_{-1}^A are given by

$$\begin{aligned} \pi c_{\pm 1}^A &= \theta_h(c^2 - d^2 \pm 2icd) \ + \ \theta_r[(b-d)^2 - (a-c)^2 \mp 2i(b-d)(a-c)]/2 \\ &\quad + \ \theta_t(a^2 - b^2 \pm 2iab) \ + \ \theta_\ell[(b+d)^2 - (a+c)^2 \mp 2i(b+d)(a+c)]/2, \end{aligned}$$

$$\begin{aligned} \pi c_0^A &= 2\theta_h(c^2+d^2) \ - \ \theta_r[(b-d)^2 + (a-c)^2] \\ &\quad + 2\theta_t(a^2+b^2) \ - \ \theta_\ell[(b+d)^2 + (a+c)^2], \end{aligned}$$

where the angles are

$$\theta_h = \tan^{-1}\left(\frac{2cb}{d^2 - c^2}\right), \quad \theta_\ell = \tan^{-1}\left(\frac{2(b+d)(a+c)}{(b+d)^2 - (a+c)^2}\right),$$

$$\theta_t = \tan^{-1}(\frac{2ab}{b^2 - a^2}), \quad \theta_r = \tan^{-1}(\frac{2(b-d)(a-c)}{(b-d)^2 - (a-c)^2}).$$

26. A method as recited in claim 22 wherein the intermediate transform is a wavelet transform based on arithmetic wavelets, and the calculation of the intermediate transform includes the step of renormalization.

27. A method as recited in claim 22 wherein the primary transformation consists of the Fourier transform, and the intermediate transform is a wavelet transformation based on modular wavelets.

28. A method as recited in claim 22 wherein the intermediate transform is a wavelet transform based on modular wavelets, and the calculation of the intermediate transform includes the step of renormalization.

29. A method as recited in claim 22 wherein the primary transformation consists of the Fourier transform, and the intermediate transform is a wavelet transformation based on fan wavelets.

30. A method as recited in claim 22 wherein the intermediate transform is a wavelet transform based on fan wavelets, and the calculation of the intermediate transform includes the step of renormalization.

31. A method as recited in claim 22 wherein the sampling of data at the points of the Farey quadrature includes temporary storage of sequences of data in a buffer.

32. A method as recited in claim 22 wherein the input data is multi-dimensional.

33. A method as recited in claim 22 wherein the constants and coefficients required in the calculation of the intermediate transform and in the calculation of the primary transform are archived and recovered from memory at run time.

34. A method of data processing comprising the steps of: providing input data in an initial form; sampling the input data at the points of the Farey quadrature; transforming the Farey-sampled data to a second form; and processing the data in the second form.

35. A method as recited in claim 34 wherein: the step of transforming the Farey-sampled data into the second form is achieved via a wavelet transformation; and the second form of the Farey-sampled data takes the form of wavelet coefficients.

36. A method as recited in claim 35 wherein the wavelet transformation is based on arithmetic wavelets.

37. A method as recited in claim 36 wherein the arithmetic wavelets are defined in accordance with the following expressions:

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \le \theta \le \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \le \theta \le \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \le \theta \le \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \le \theta \le 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetic arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

38. A method as recited in claim 35 wherein the wavelet transformation is based on modular wavelets.

39. A method as recited in claim 38 wherein the modular wavelets are defined in accordance with the following expressions:

$$\psi_A(\theta) = \sum_{n\ge 0} n\ \overline{\vartheta}_{U^n A}(\theta),$$

$$\psi_{SA}(\theta) = \sum_{n\ge 0} n\ \overline{\vartheta}_{T^{-n} A}(\theta),$$

where

$$\overline{\vartheta}_A(\theta) = \begin{cases} \tilde{\vartheta}_A(\theta); & \text{if } A \ne U^{-1}, T^{-1}, \\ \tilde{\vartheta}_{U^{-1}}(\theta) - 2\sin\theta; & \text{if } A = U^{-1}, \\ \tilde{\vartheta}_{T^{-1}}(\theta) + 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $\cdot = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$; and

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \le \theta \le \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \le \theta \le \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \le \theta \le \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \le \theta \le 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetric arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

40. A method as recited in claim 35 wherein the wavelet transformation is based on fan wavelets.

41. A method as recited in claim 40 wherein the fan wavelets are defined in accordance with the following expressions:

$$\phi_A(\theta) = \sum_{n\ge 0} \overline{\vartheta}_{U^n A}(\theta),$$

$$\phi_{SA}(\theta) = \sum_{n\ge 0} \overline{\vartheta}_{T^{-n} A}(\theta),$$

where

$$\overline{\vartheta}_A(\theta) = \begin{cases} \tilde{\vartheta}_A(\theta); & \text{if } A \ne U^{-1}, T^{-1}, \\ \tilde{\vartheta}_{U^{-1}}(\theta) - 2\sin\theta; & \text{if } A = U^{-1}, \\ \tilde{\vartheta}_{T^{-1}}(\theta) + 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $L = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$; and

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \le \theta \le \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \le \theta \le \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \le \theta \le \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \le \theta \le 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetic arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

42. A method as recited in claim 35 wherein the steps of sampling the input data at the points of the Farey quadrature and transforming the Farey-sampled into wavelet coefficients is characterized by a binary cascade of arrow structures arising from the calculation of wavelet coefficients by sampling the input data in its initial form at the points of the Farey quadrature as defined along a circle in the complex plane.

43. A method as recited in claim 42 wherein the wavelet transformation is based on arithmetic wavelets and the step of transforming the Farey-sampled data into arithmetic wavelet coefficients includes the step of regularization.

44. A method as recited in claim 35 wherein the wavelet transformation is based on a finite number of wavelets.

45. A method as recited in claim 35 wherein: wavelet transformation is achieved via a plurality of wavelet transformation operations each for a specified arc along a circle in

the complex plane; and e plurality of wavelet transformation oper ons are calculated in parallel with each other.

46. A method as recited in claim 34 wherein the input data is digital data.

47. A method as recited in claim 34 wherein the input data is analog date and the analog input data is sampled by placing analog sensors at locations corresponding to the points of the Farey quadrature.

48. A method as recited in claim 34 wherein the input data is multi-dimensional.

49. A method as recited in claim 34 wherein the sampling of data at the points of the Farey quadrature includes temporary storage of sequences of data in a buffer.

50. A digital processing system comprising a digital filter that receives an input signal and outputs a transformed signal, the digital filter being an arithmetic wavelet filter in which the arithmetic wavelets are defined in accordance with the following expressions:

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \le \theta \le \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \le \theta \le \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \le \theta \le \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \le \theta \le 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetric arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

51. A digital processing system comprising a digital filter that receives an input signal and outputs a transformed signal, the digital filter being a modular wavelet filter in which the modular wavelets are defined in accordance with the following expressions:

$$\psi_A(\theta) = \sum_{n\geq 0} n\, \overline{\vartheta}_{U^n A}(\theta),$$
$$\psi_{SA}(\theta) = \sum_{n\geq 0} n\, \overline{\vartheta}_{T^{-n} A}(\theta),$$

where

$$\overline{\vartheta}_A(\theta) = \begin{cases} \tilde{\vartheta}_A(\theta); & \text{if } A \neq U^{-1}, T^{-1}, \\ \tilde{\vartheta}_{U^{-1}}(\theta) - 2\sin\theta; & \text{if } A = U^{-1}, \\ \tilde{\vartheta}_{T^{-1}}(\theta) + 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$; and

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \leq \theta \leq \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \leq \theta \leq \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \leq \theta \leq \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \leq \theta \leq 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetic arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \leq \theta \leq \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \leq \theta \leq \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \leq \theta \leq \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \geq a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

52. A digital processing system comprising a digital filter that receives an input signal and outputs a transformed signal, the digital filter being a fan wavelet filter in which the fan wavelets are defined in accordance with the following expressions:

$$\phi_A(\theta) = \sum_{n\geq 0} \overline{\vartheta}_{U^n A}(\theta),$$
$$\phi_{SA}(\theta) = \sum_{n\geq 0} \overline{\vartheta}_{T^{-n} A}(\theta),$$

where

$$\overline{\vartheta}_A(\theta) = \begin{cases} \tilde{\vartheta}_A(\theta); & \text{if } A \neq U^{-1}, T^{-1}, \\ \tilde{\vartheta}_{U^{-1}}(\theta) - 2\sin\theta; & \text{if } A = U^{-1}, \\ \tilde{\vartheta}_{T^{-1}}(\theta) + 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$; and

$$\tilde{\vartheta}_I(\theta) = \begin{cases} +2\cos\theta + 2\sin\theta - 2; & \text{if } 0 \leq \theta \leq \frac{\pi}{2}, \\ +2\cos\theta - 2\sin\theta + 2; & \text{if } \frac{\pi}{2} \leq \theta \leq \pi, \\ -2\cos\theta - 2\sin\theta - 2; & \text{if } \pi \leq \theta \leq \frac{3\pi}{2}, \\ -2\cos\theta + 2\sin\theta + 2; & \text{if } \frac{3\pi}{2} \leq \theta \leq 2\pi; \end{cases}$$

and if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$ labels an arithmetic arrow;

and if A labels a top arithmetic arrow and $a > c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(d^2-c^2)\cos\theta + 8cd\sin\theta - 4(d^2+c^2); \\ \qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \leq \theta \leq \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +2(a^2+d^2-b^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(cd-ab-bc-ad)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \leq \theta \leq \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+d^2-a^2-c^2+2ac-2bd)\cos\theta \\ \quad +4(ab+cd-ad-bc)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2+2ac+2bd); \\ \qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \leq \theta \leq \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ 0; \qquad \text{otherwise;} \end{cases}$$

and if A labels a top arithmetic arrow and $c \geq a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2-2ac+2bd)\cos\theta \\ \quad +4(ad+bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(a^2+d^2-b^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd-ab)\sin\theta \\ \quad +2(a^2+b^2-c^2-d^2-2bd-2ac); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}, \\ +4(a^2-b^2)\cos\theta - 8ab\sin\theta + 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $a > -c$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +2(a^2+c^2-b^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(-ad-bc-ab-cd)\sin\theta \\ \quad +2(a^2+b^2+c^2+d^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2} \le \theta \le \tan^{-1}\frac{2ba}{b^2-a^2}, \\ +2(b^2+c^2-a^2-d^2+2ac-2bd)\cos\theta \\ \quad +4(ab-ad-bc-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2+2ac+2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +4(c^2-d^2)\cos\theta - 8cd\sin\theta + 4(c^2+d^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ 0; \qquad\qquad \text{otherwise;} \end{cases}$$

and if A labels a bottom arithmetic arrow and $-c \ge a$, then

$$\tilde{\vartheta}_A(\theta) = \begin{cases} +4(b^2-a^2)\cos\theta + 8ab\sin\theta - 4(a^2+b^2); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2ba}{b^2-a^2} \le \theta \le \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2}, \\ +2(b^2+c^2-a^2-d^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+ab-cd)\sin\theta \\ \quad +2(c^2+d^2-a^2-b^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2(d-b)(c-a)}{(d-b)^2-(c-a)^2} \le \theta \le \tan^{-1}\frac{2cd}{d^2-c^2}, \\ +2(b^2+d^2-a^2-c^2+2bd-2ac)\cos\theta \\ \quad +4(ad+bc+cd+ab)\sin\theta \\ \quad +2(-a^2-b^2-c^2-d^2-2ac-2bd); \\ \qquad\qquad \text{if } \tan^{-1}\frac{2cd}{d^2-c^2} \le \theta \le \tan^{-1}\frac{2(b+d)(a+c)}{(b+d)^2-(a+c)^2}. \\ 0; \qquad\qquad \text{otherwise.} \end{cases}$$

53. A method of obtai .g an inverse transform of input digital a comprising the steps of: providing input digital data in an initial form; transforming the data in the initial form to an intermediate form; and transforming the intermediate form of the data using an intermediate inverse transform to obtain output values at the points of the Farey quadrature.

54. A method as recited in claim 53 wherein the input digital data is in the form of Fourier coefficients and the combination of transforming the data from the Fourier coefficients to the intermediate form and from the intermediate form to obtain output values at the points of the Farey quadrature constitutes an inverse Fourier transform.

55. A method as recited in claim 54 wherein the intermediate form of the data is in the form of wavelet coefficients.

56. A method as recited in claim 55 wherein the wavelet coefficients are coefficients for arithmetic wavelets,

$$\tilde{\vartheta}_A(\theta) = \begin{cases} \overline{\vartheta}_A(\theta); & \text{if } A \neq U^{-1}, T^{-1}, \\ \overline{\vartheta}_{U^{-1}}(\theta) + 2\sin\theta; & \text{if } A = U^{-1}, \\ \overline{\vartheta}_{T^{-1}}(\theta) - 2\sin\theta; & \text{if } A = T^{-1}, \end{cases}$$

with $U^{-1} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$, and $T^{-1} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$, and conversion of the data from the initial form to the intermediate form is based on the following expressions:

$$\begin{aligned} e^{in\theta} &= \cos n\theta + i \sin n\theta \\ &= -\left[c_0^n + c_1^n e^{i\theta} + c_{-1}^n e^{-i\theta}\right] + \frac{i}{4} \sum_A \left\{ n(\xi^n + \eta^n) + \frac{\eta + \xi}{\eta - \xi} (\xi^n - \eta^n) \right\} \overline{\vartheta}_A(\theta), \end{aligned}$$

where the sum is over all arithmetic arrows, the underlying chord of which has complex endpoints ξ, η, and

$$c_0^n = \begin{cases} -1, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -1, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_1^n = \begin{cases} 0, & n \equiv 0(4); \\ -1, & n \equiv 1(4); \\ i, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_{-1}^n = \begin{cases} 0, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -i, & n \equiv 2(4); \\ -1, & n \equiv 3(4). \end{cases}$$

57. A method as recited in claim 55 wherein the wavelet coefficients are coefficients for modular wavelets, ψ_A, and conversion of the data from the initial form to the intermediate form is based on the following expressions:

$$\begin{aligned} e^{in\theta} &= \cos n\theta + i \sin n\theta \\ &= -\left[c_0^n + c_1^n e^{i\theta} + c_{-1}^n e^{-i\theta}\right] \\ &\quad + \frac{i}{4} \sum_A \left\{ n(\xi^n + \eta^n) + \frac{\eta + \xi}{\eta - \xi} (\xi^n - \eta^n) \right\} \left[\psi_{UA}(\theta) - 2\psi_A(\theta) + \psi_{U^{-1}A}(\theta)\right], \end{aligned}$$

where the sum is over a arithmetic arrows, the underlying chord of which has complex endpoints ξ, η, where $U^{\pm 1} = \begin{pmatrix} 1 & 0 \\ \pm 1 & 1 \end{pmatrix}$, and where

$$c_0^n = \begin{cases} -1, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -1, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_1^n = \begin{cases} 0, & n \equiv 0(4); \\ -1, & n \equiv 1(4); \\ i, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_{-1}^n = \begin{cases} 0, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -i, & n \equiv 2(4); \\ -1, & n \equiv 3(4). \end{cases}$$

58. A method as recited in claim 55 wherein the wavelet coefficients are coefficients for fan wavelets, ϕ_A, and conversion of the data from the initial form to the intermediate form is based on the following expressions:

$$\begin{aligned} e^{in\theta} &= \cos n\theta + i \sin n\theta \\ &= -\left[c_0^n + c_1^n e^{i\theta} + c_{-1}^n e^{-i\theta}\right] \\ &\quad + \frac{i}{4} \sum_A \left\{ n(\xi^n + \eta^n) + \frac{\eta + \xi}{\eta - \xi} (\xi^n - \eta^n) \right\} \left[\phi_A(\theta) - \phi_{UA}(\theta)\right], \end{aligned}$$

where the sum is over all arithmetic arrows, the underlying chord of which has complex endpoints ξ, η, where $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and where

$$c_0^n = \begin{cases} -1, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -1, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_1^n = \begin{cases} 0, & n \equiv 0(4); \\ -1, & n \equiv 1(4); \\ i, & n \equiv 2(4); \\ 0, & n \equiv 3(4); \end{cases} \quad c_{-1}^n = \begin{cases} 0, & n \equiv 0(4); \\ 0, & n \equiv 1(4); \\ -i, & n \equiv 2(4); \\ -1, & n \equiv 3(4). \end{cases}$$

59. A method as recited in claim 53 wherein the intermediate form of the data is in the form of wavelet coefficients.

60. A method as recited in claim 53 wherein the initial form of the input digital data is Fourier coefficients, the intermediate form of the data is in the form of wavelet coefficients, and the output values at the points of the Farey quadrature are obtained via the use of a binary cascade of arrow structures arising from the calculation of wavelet coefficients as defined along a circle in the complex plane.

61. A method as recited in claim 60 further comprising the step of storing terminal arrow structures in order for restart or iterative refinement.

62. A method as recited in claim 59 wherein the wavelet coefficients are coefficients for arithmetic wavelets.

63. A method as recited in claim 59 wherein the wavelet coefficients are coefficients for modular wavelets.

64. A method as recited in claim 59 wherein the wavelet coefficients are coefficients for fan wavelets.

65. A method as recited in claim 59 wherein the input data is multi-dimensional.

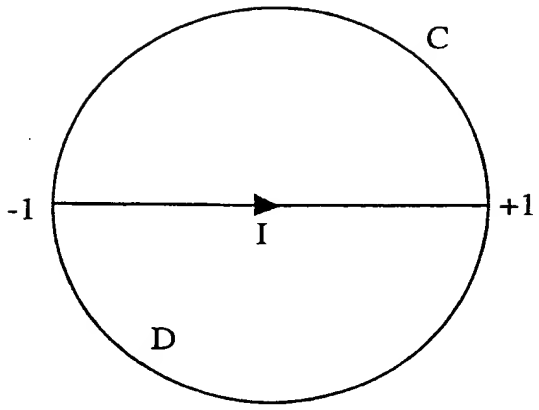


FIGURE 1a

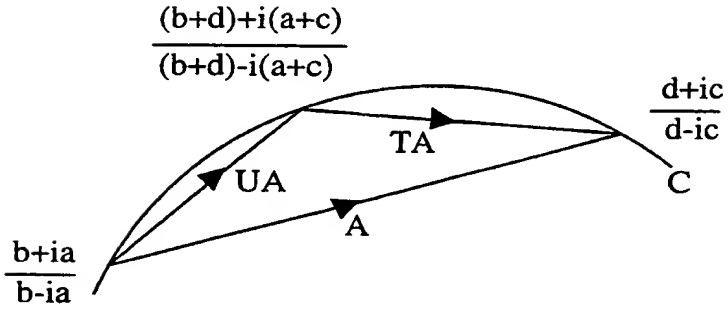


FIGURE 1b

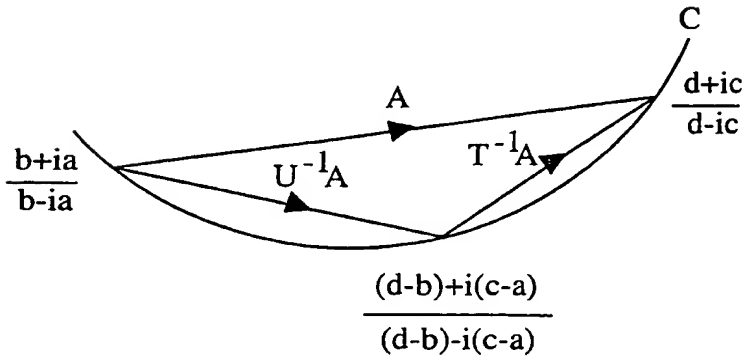


FIGURE 1c

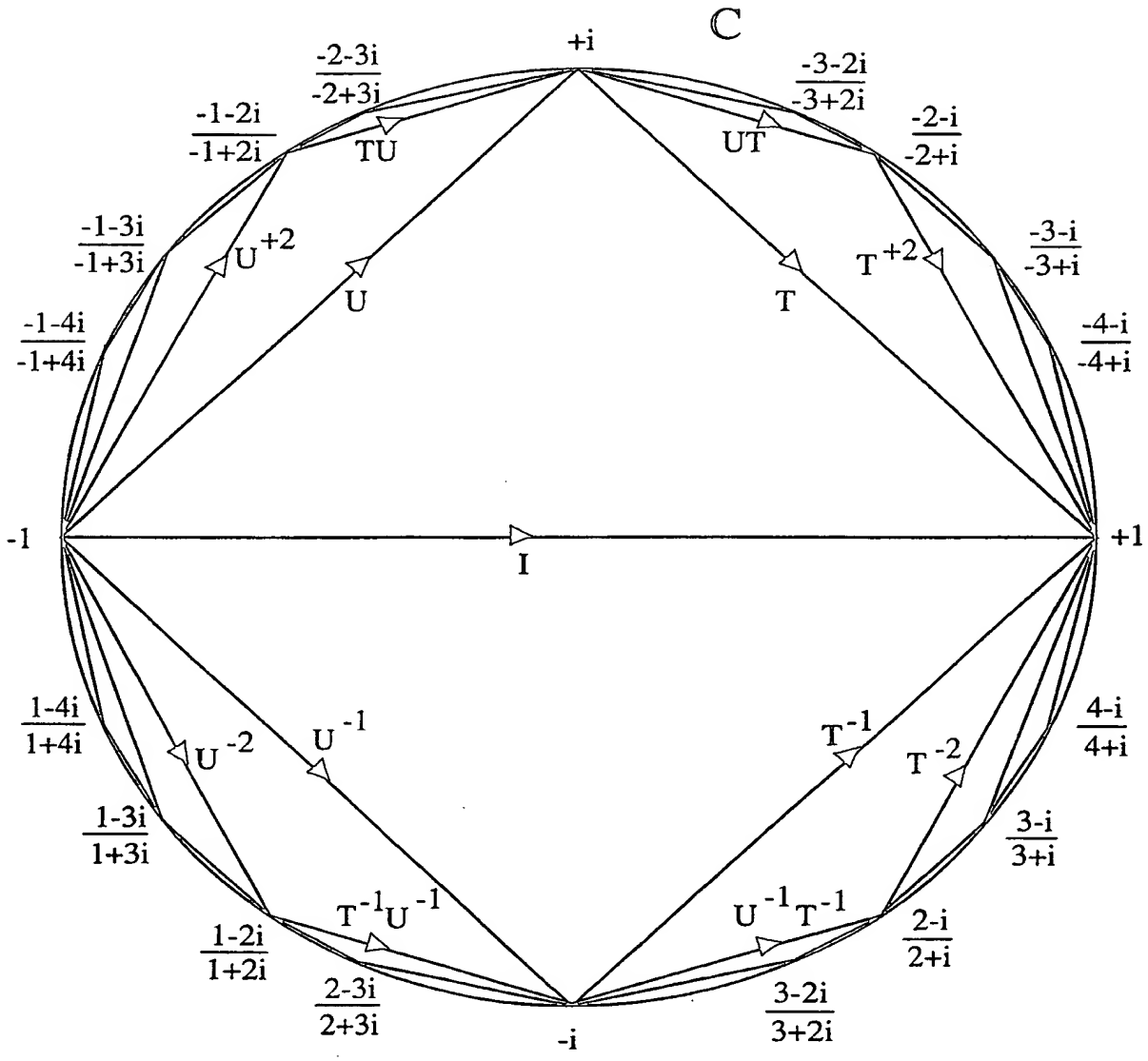

C
+i
-2-3i / -2+3i
-3-2i / -3+2i
-1-2i / -1+2i
-2-i / -2+i
TU
UT
-1-3i / -1+3i
-3-i / -3+i
U+2
T+2
U
T
-1-4i / -1+4i
-4-i / -4+i
-1
+1
I
1-4i / 1+4i
4-i / 4+i
U-2
U-1
T-1
T-2
1-3i / 1+3i
3-i / 3+i
T-1U-1
U-1T-1
1-2i / 1+2i
2-i / 2+i
2-3i / 2+3i
3-2i / 3+2i
-i

FIGURE 2

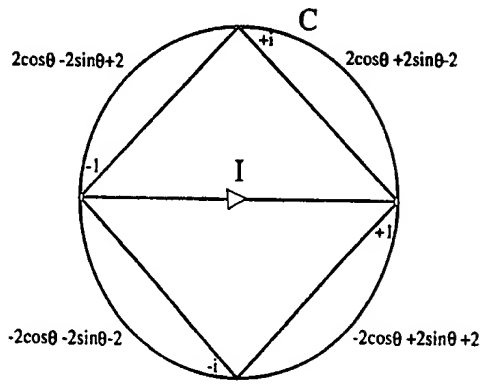


FIGURE 3a

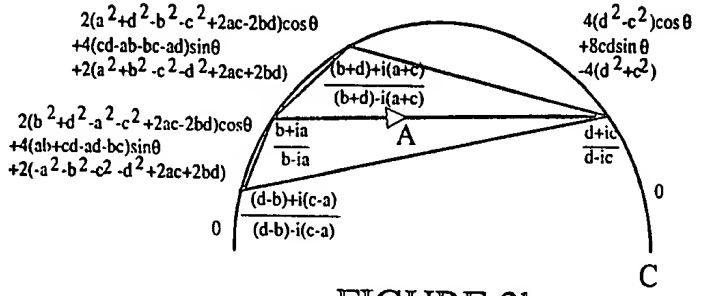


FIGURE 3b

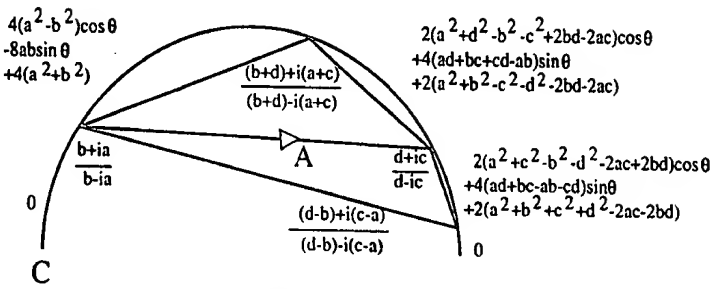


FIGURE 3c

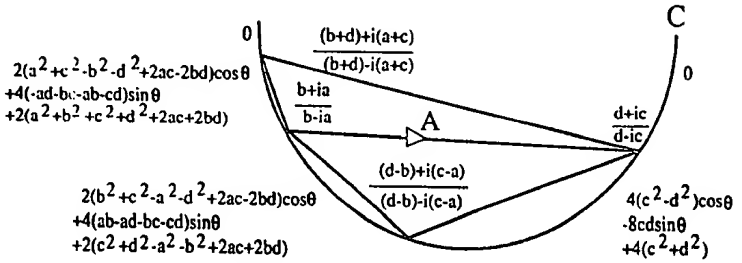


FIGURE 3d

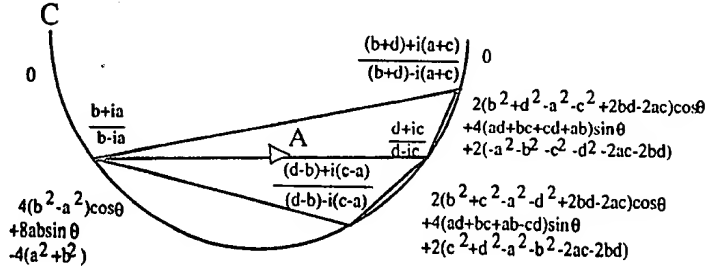


FIGURE 3e

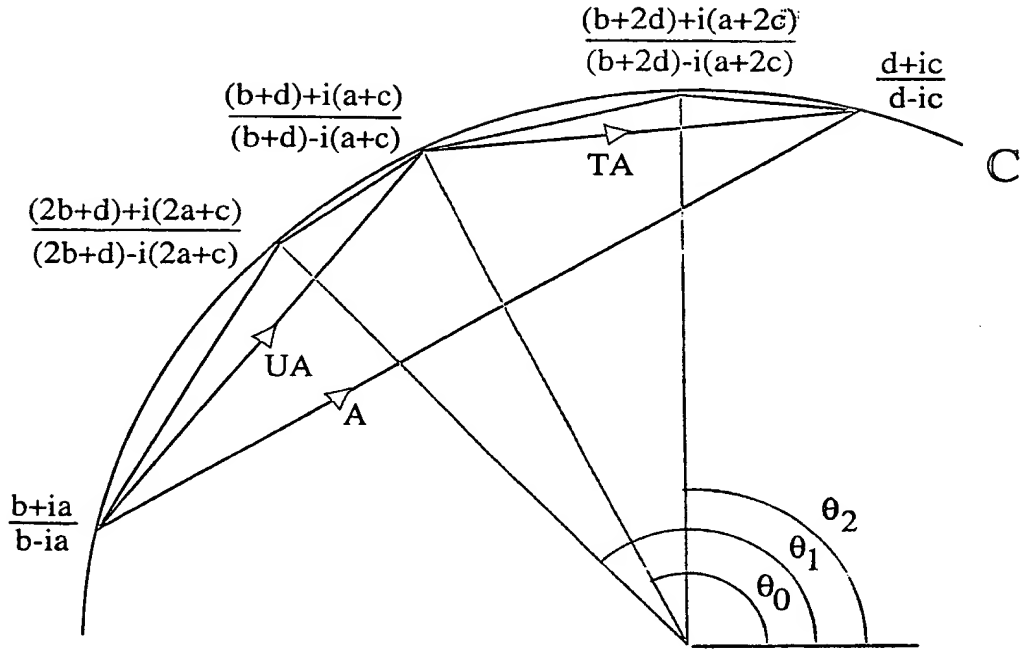


FIGURE 4

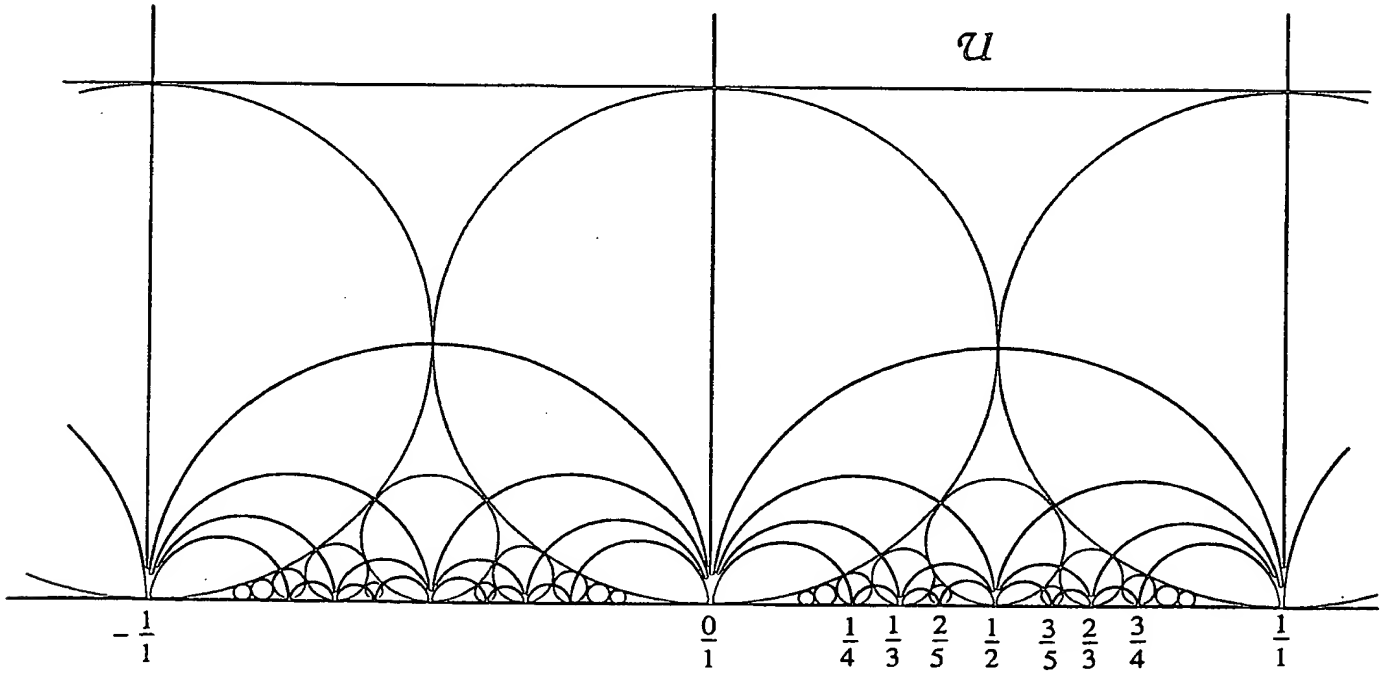


FIGURE 11

Routine: main

Data, Variables, and Effects:

input: data presented as a function f(p,q), giving the function value at the point (p-iq)/(p+iq)
output: array of complex Fourier coefficients
effect: calculates Fourier transform

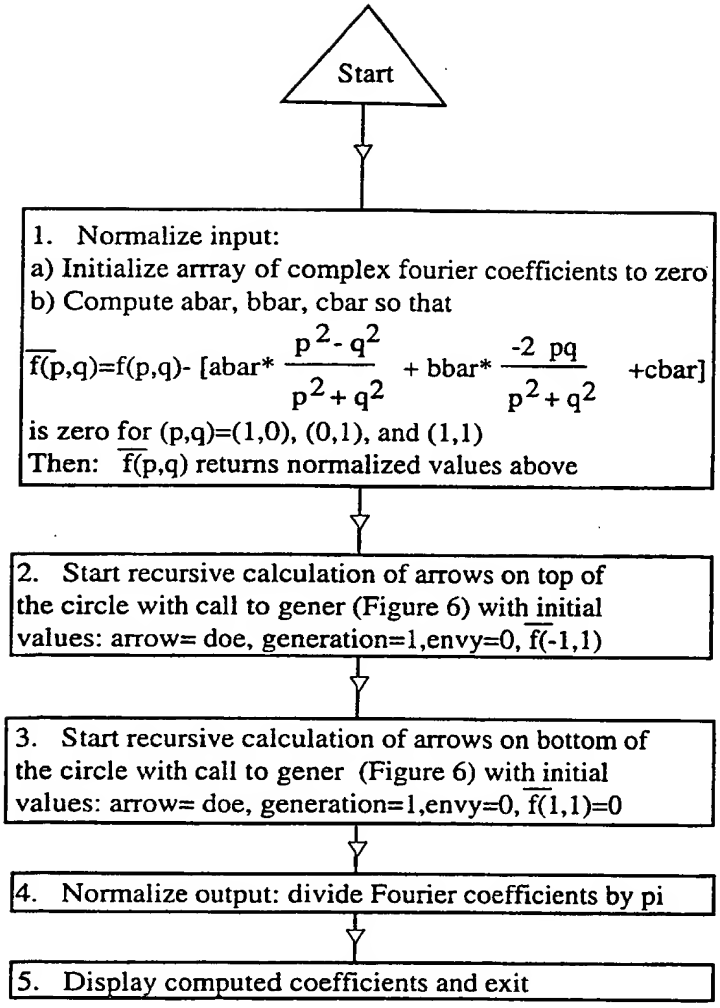
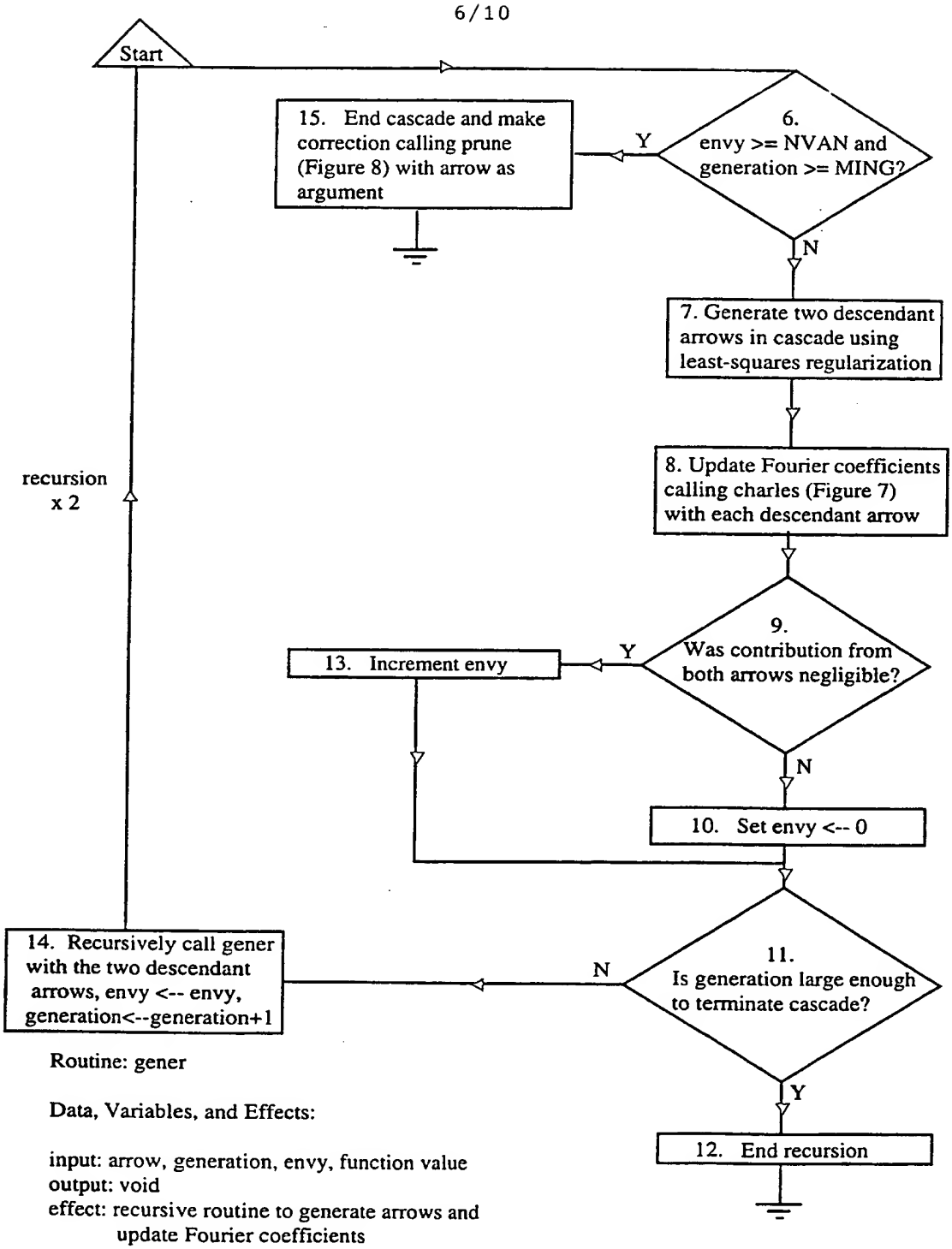


FIGURE 5



Routine: gener

Data, Variables, and Effects:

input: arrow, generation, envy, function value
output: void
effect: recursive routine to generate arrows and update Fourier coefficients

FIGURE 6

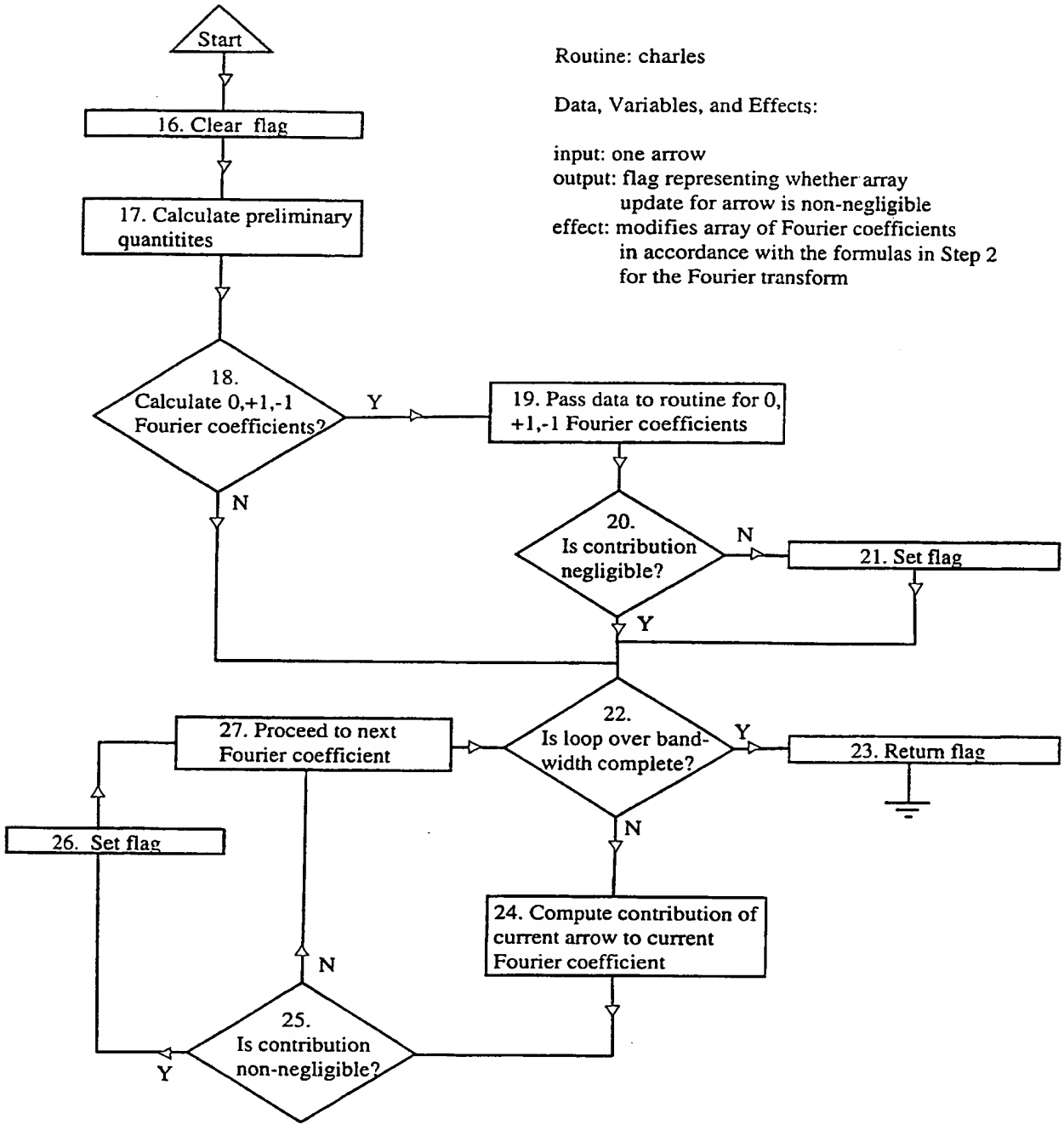
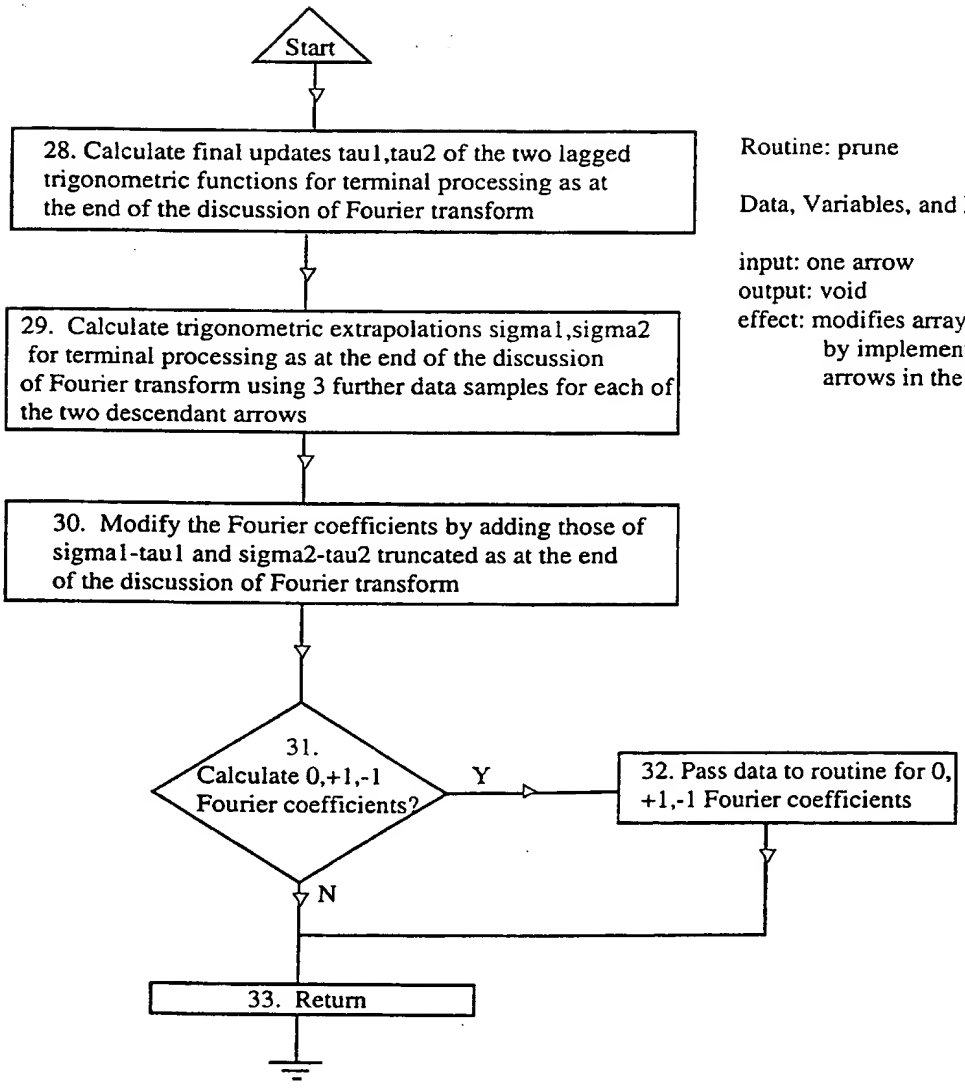

Start
16. Clear flag
17. Calculate preliminary quantitites
18. Calculate 0,+1,-1 Fourier coefficients?
Y
19. Pass data to routine for 0, +1,-1 Fourier coefficients
N
20. Is contribution negligible?
N
21. Set flag
Y
22. Is loop over band-width complete?
Y
23. Return flag
N
27. Proceed to next Fourier coefficient
26. Set flag
24. Compute contribution of current arrow to current Fourier coefficient
N
25. Is contribution non-negligible?
Y
Routine: charles
Data, Variables, and Effects:
input: one arrow
output: flag representing whether array update for arrow is non-negligible
effect: modifies array of Fourier coefficients in accordance with the formulas in Step 2 for the Fourier transform

FIGURE 7

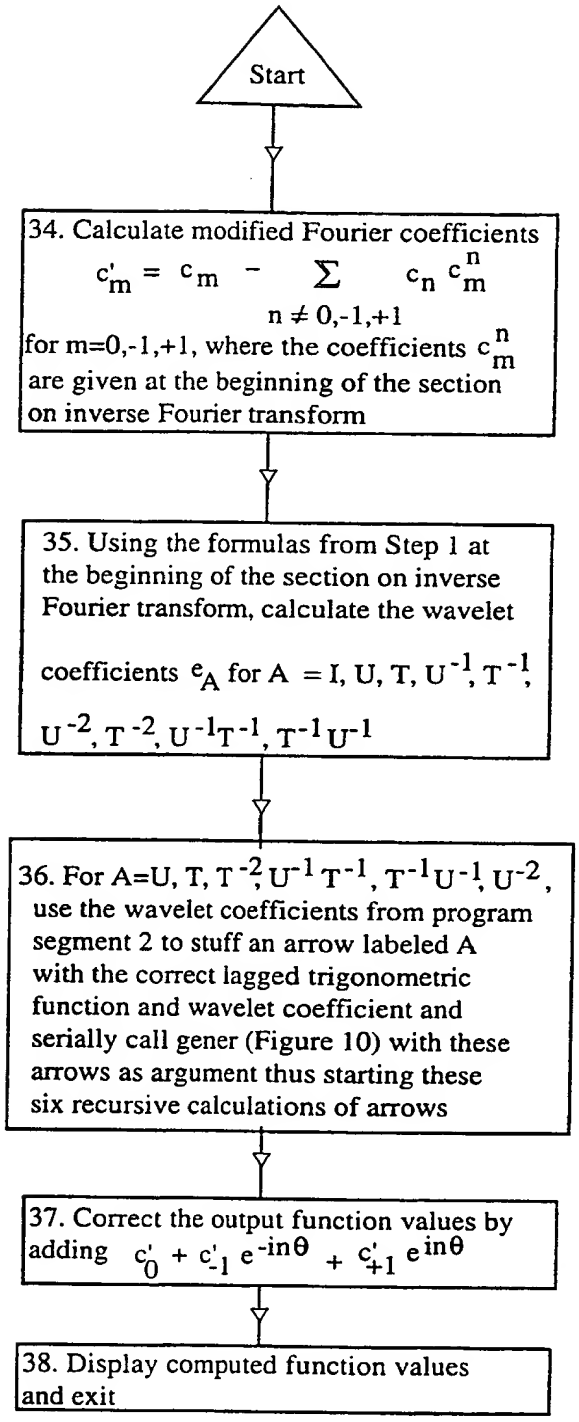


Routine: prune

Data, Variables, and Effects:

input: one arrow
output: void
effect: modifies array of Fourier coefficients by implementing the final processing of arrows in the cascade

FIGURE 8

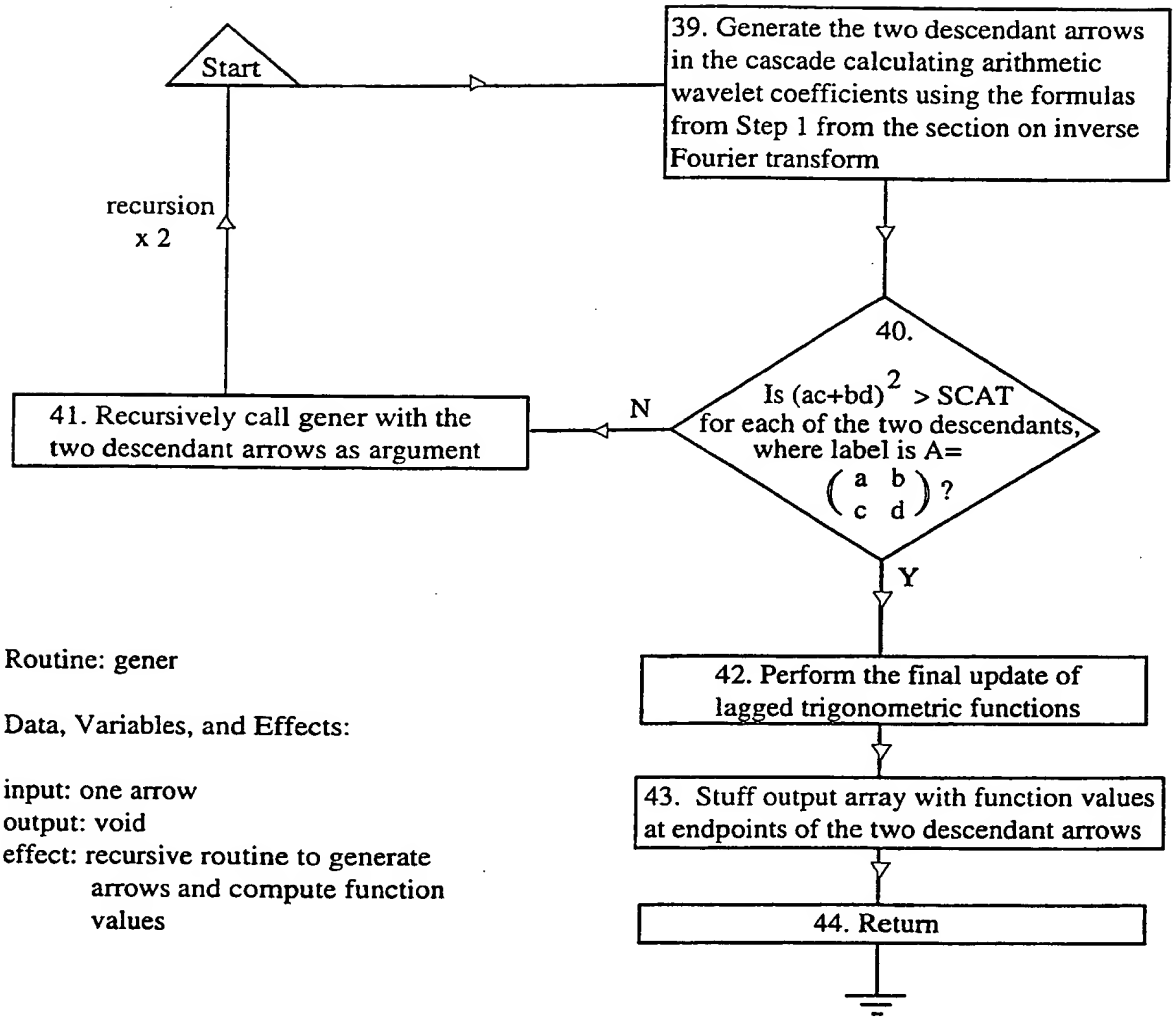


Routine: main

Data, Variables, Effects:

input: array of complex Fourier coefficients
c for |n| <= N
output: array of three-tuples (p,q,z), where output function takes complex value z at (p-iq)/(p+iq) given in the counter-clockwise order on the circle
effect: calculates inverse Fourier transform

FIGURE 9



Routine: gener

Data, Variables, and Effects:

input: one arrow
output: void
effect: recursive routine to generate arrows and compute function values

FIGURE 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/30584

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) :H04B 1/66; G06K 9/36, 9/46
US CL : 375/240; 382/232, 276
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 375/240, 241; 382/232, 248, 251, 276; 364/725.01, 726.01; 704/230, 503

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST, IEL

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,802,481 A (PRIETO) 01 September 1998, all	1-65
A	US 5,819,215 A (DOBSON et al) 06 October 1998, all	1-65
A, P	US 5,974,181 A (PRIETO) 26 October 1999, all	1-65
A, P	US 5,991,454 A (FOWLER) 23 November 1999, all	1-65

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
10 MARCH 2000

Date of mailing of the international search report
12 APR 2000

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
STEPHEN CHIN
Telephone No. (703) 305-4714

Form PCT/ISA/210 (second sheet)(July 1992) ☆